AN IMPROVED ALGORITHM FOR TRANSLATING RELATIONAL SCHEMAS INTO AN OBJECT MODEL

THESIS

Joseph C. Pearson, Captain, USAF

AFIT/GCS/ENG/00J-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/00J-04

AN IMPROVED ALGORITHM FOR TRANSLATING RELATIONAL SCHEMAS INTO AN OBJECT MODEL

THESIS

Joseph C. Pearson, Captain, USAF

AFIT/GCS/ENG/00J-04

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense or the U. S. Government.

AFIT/GCS/ENG/00J-04

AN IMPROVED ALGORITHM FOR TRANSLATING RELATIONAL SCHEMAS INTO AN OBJECT MODEL

THESIS

Presented to the Faculty of the Graduate School of Engineering and Management

of the Air Force Institute of Technology

Air University in Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Systems

Joseph C. Pearson, B.S.

Captain, USAF

June 2000

AFIT/GCS/ENG/00J-04

AN IMPROVED ALGORITHM FOR TRANSLATING RELATIONAL SCHEMAS INTO AN OBJECT MODEL

Joseph C. Pearson. B.S.

Captain, USAF

Approved:

Thomas C. Hartrum, Ph.D.
Committee Chairman — date: 25 May 2000

Scott A. DeLoach, Ph.D., Major
Committee Member — date: 25 May 2000

Michael L. Talbert, Ph.D., Major
Committee Member — date: 25 May 2000

Acknowledgments

I'm grateful to my thesis advisor, Dr. Hartrum, for his patience, guidance and for not giving up on me when he must have been sorely tempted. Thank you, professor, committee member and mentor, Major Talbert. You will be long remembered for the sage advice, perspective on matters technical and personal, and the example you set. Thanks, Major DeLoach, committee member and professor, for asserting that there is other artificial intelligence besides mine. Thank you Dr. Potoczny, professor and "Ace Geometer," for making all subjects interesting, challenging and fun--anyone who graduates from AFIT without taking at least one of your classes is missing out.

More fellow students had a positive influence on my AFIT experience than can be cited, but some notables are Merriellen Joga, Tim Lacey, Dale Lathrop, David Marsh, and Mark Wood. I'm grateful for Steven Buckwalter and his wonderful wife Patty who shared much. Our Aussie student, Steven Thomson made my final quarter after the majority of the class graduated a far more enjoyable experience than it would've otherwise been.

I owe huge thanks to lifelong friends Juan and Teresa Odenwood, and to my family that has quality and good Irish disposition to match our quantity. Love and thanks to my three girls Antraneva, Charlene and Monica, who are the reason the effort is worthwhile and who make me a better man than my nature disposes me to be. Finally, thanks to He who allows me yet another opportunity to live and learn.

JoE Pearson

Table of Contents

Abstract

Today's warfighter is inundated with data from numerous Command, Control, Communications and Computers and Intelligence systems. Integration of these systems is desirable, yet integration results in a static solution to a dynamic problem—by the time a global schema can be devised, it's out of date. Automating schema integration will mitigate this problem, but data model disparity must be addressed via translation to a common data model prior to integration. To address this requirement, this thesis presents an improved, relational to object-oriented schema translation algorithm, which is derived from a base algorithm proposed by another research effort. The improved algorithm incorporates many benefits over the base algorithm, including increased automation, semantic equivalence assurance via tracing key migrations, and decreased inter-entity association redundancy. The improved algorithm is implemented in a highly automated schema translation application, which is demonstrated against a sample schema and validated on another schema from an operational course scheduling system. Given a relational schema, the application provides an initial object-oriented translation and re-translates when schema change is detected.

An Improved Algorithm for Translating Relational Schemas into an Object Model

I. Introduction

1.1 Background

Today's warfighter is inundated with data from numerous Command, Control, Communications and Computers and Intelligence (C4I) systems. Aggregation of decision support tools and attendant data sources into fewer systems is desirable, but interoperability concerns and a variety of technical challenges are formidable barriers to resolving this data-management dilemma. Integrating distributed, heterogeneous systems is a well-studied problem, with solutions ranging from a centralized database accessible to the integrated systems to inter-system data conversion implemented via software with decentralized data storage. A significant difficulty with these approaches is that they inevitably result in a static solution to a dynamic problem--by the time a global schema can be devised and implemented, the solution is out of date. Another major concern is the prohibitive expense of continuously updating the schema to accommodate needed changes. These problems beg for dynamic solutions, which have yet to be devised.

1.2 Problem

The goal of this research was to determine the feasibility of automating the translation of heterogeneous data source schemas into a common object model to facilitate integration. A characterization of the barriers to automating translation tasks and a scaleable, extensible methodology to address diverse data source schemas were to be provided.

1.3 Current Initiatives

Schema integration is a well-studied problem, yet efforts to automate the integration and schema translation are immature. Ashby [20] proposes a partially automated schema integration methodology, but like most approaches, it is scoped to accept schemas that are already in a common data model.

1.4 Scope

An important assumption of the integration methodology proposed by Ashby is that object-oriented schemas are the input to a schema integration effort, which precludes his methodology from having to address data model conflicts during integration. This research effort addresses automation of the schema translation effort that must precede any integration effort addressed by Ashby, or similar integration methodologies. The schema translation effort facilitates, but does not directly address, schema integration, query processing, and transaction management. Automation of various schema translation tasks,

schema change detection, and object model re-translation are presented, and an analysis of tasks not automated is provided.

1.5 Contributions

This effort identifies the tasks required to translate relational schemas into a common object model. Automation of these tasks was attempted with varying success, and a characterization of translation tasks with respect to automation is provided. Automated schema change detection and re-translation schemes were devised and evaluated. A prototype relational to object-oriented schema translation application was implemented to automate translation, capture design decisions, and automatically propagate updates to the object-oriented schema. Simple instructions to apply this application against any relational schema residing on an ODBC compatible DBMS are provided. Finally, a general methodology for extending the automated schema translation application to address diverse schema types is proposed.

1.6 Document Organization

Chapter 2, Background, begins with an overview of multi-database management systems and schema integration, continues with a discussion of current research initiatives in schema integration, and concludes with a description of a promising schema integration methodology proposed by Sull and Kashyap [5], which provided the base schema translation algorithm used in this research. Chapter 3, Methodology, provides the plan

followed in this research to create a portable and extensible automated schema translation application. Chapter 4, Results and Analysis, describes the implementation of the translation application introduced in Chapter 3 and provides an analysis of schema translation tasks with respect to automation. Finally, Chapter 5, Conclusions and Recommendations, provides a summary of research findings and proposes areas for further study.

II. Background

This chapter begins with an introduction to solutions that range from global schema integration to federated databases and multi database languages. Issues associated with various Multi Database Management System solutions are discussed. Next, the schema integration process is described, and various approaches are contrasted.

2.1 Multi Database Management Systems (MDBS)

An MDBS resides unobtrusively atop existing, independently designed component DBMS and file systems to present the illusion of a single DBMS to MDBS users. This is accomplished via a global schema that neutralizes conflicts between the local database system (LDBS) schemas. Global queries and sometimes updates are dispatched to and merged from LDBSs, and transaction management is coordinated among LDBSs by the MDBS.[7:516]

According to Kim [7], the general objectives of an MDBS are:

- It must not require migration from one data source to another, e.g. from ORACLE to SYBASE.
- It must not require changes to LDBSs in order to operate.
- It must not prevent LDBSs from being used independently from the MDBS. Data managed by the LDBS must be accessible directly via the LDBS, and data managed by multiple LDBSs must be accessible via the MDBS.

- It must provide a common language that precludes users from interfacing via multiple LDBS languages.
- It must resolve LDBS heterogeneity.
- It must support distributed transactions including updates across LDBSs.
- It must provide the standard facilities expected of a "full-blown DBMS."
- It must not cause significant change to the operation and administration of LDBSs.
- It must have similar performance to a homogeneous, distributed DBMS; i.e. heterogeneity resolution mustn't be prohibitively costly.
- It must allow uniform access to all data and resources, and provide cooperation via data exchange and synchronized execution [1].

2.1.1 MDBS Issues

2.1.1.1 Autonomy

Autonomy[7,1], a key concept in MDBSs, has the following variants:

- *Design autonomy* – there's no need to modify LDBS software to accommodate the MDBS. LDBSs can have heterogeneous implementations, e.g. data model, query language, etc.
- *Execution Autonomy* –each LDBS maintains complete control over transactions it executes.
- *Communications autonomy* – LDBSs don't share control information with each other or the MDBS.
- *Association autonomy* – each LDBS decides what functions, operations, data and metadata to share.

2.1.1.2 Semantic Relationships

A concept can have different representations in different schemas and the following *semantic relationships* can exist between representations R1 and R2 [8]:

- *Identical*—R1 and R2 are exactly the same using the same modeling constructs.
- *Equivalent*—R1 and R2 are exactly the same, but different, equivalent modeling constructs are used. The three types of equivalence are:
    - *Behavioral*—Every instance of R1 has a corresponding instance of R2 that provides the same query result for any given query, and vice versa.
    - *Mapping*—A one-to-one correspondence exists between instances of R1 and R2.
    - *Transformational (Restructure)*—A set of atomic transformations applied to R1 will produce R2.
- *Compatible*—R1 and R2 aren't identical or equivalent, but modeling constructs, designer perception and integrity constraints aren't violated.
- *Incompatible*—R1 and R2 are contradictory due to specification incoherence.

2.1.1.3 Heterogeneity

Representational heterogeneity refers to variations in data specification and structure between databases. The spectrum of heterogeneity can be organized along the following levels of abstraction [1]:

- *Metadata language* (conceptual database model)—LDBSs may have different data models and/or DDLs.
- *Metadata specification* (conceptual schema)—even with a common data model, the data may be specified differently within the model.

- *Object comparability*—with a common conceptual model and specification, which objects are related?
- *Low-level data format*—what units of measure, field lengths, etc. are used to represent the data?
- *Tool* (DBMS)—different tools can be used to manage and interface with data, independent of the above dimensions.

Representational heterogeneity has three main causes: [8]

- *Different perspectives and needs*—a concept may be modeled in a variety of ways by different designers who can have as many perspectives of the universe of discourse as there are designers.
- *Equivalent constructs*—the constructs available in the data model allow multiple equivalent representations for the same data. In general, the richer the data model, the more possible equivalent representations exist.
- *Incompatible design specifications*—different relationship cardinalities, etc. in design specifications result in different schemas.

2.1.1.4 Implications

From the LDBS perspective, local autonomy and heterogeneity allow local control of data and the ability to modify, manage and maintain data in a way most beneficial to each LDBS without concern for impact to an MDBS. From the MDBS perspective, local autonomy and heterogeneity degrade global consistency and efficiency, with the following implications to MDBS query optimization: [7]

- Local entity set information useful for optimization may be unavailable.
- LDBSs may use different query processing algorithms and performance measures.

- Data transmission between LDBSs may not be fully supported.

A simple data model eases schema conforming and merging because type conflicts are reduced, transformation operations are simpler and fewer primitive operations are involved in schema merging. However, data models with rich type and abstraction mechanisms allow schema comparison at higher levels of abstraction to ease discovery of similarities, dissimilarities and incompatibilities [8].

2.1.2 MDBMS Approaches

2.1.2.1 Global Schema Integration[1]

Early distributed/heterogeneity resolution efforts focused on fusing component databases at the schema level. The primary benefit of this approach is a consistent, uniform view of and access to data. The disadvantages include:

- Automating schema integration is problematic due to difficulty identifying relationships among attributes of two schemas and relationships between entity types and relationship types. In fact, the problem of relational schema integration has proven to have no general solution, therefore human intervention is required to resolve semantic, structural and behavioral conflicts.
- Semantic conflict resolution is achieved by sacrificing autonomy, and local schemas may need modification prior to integration to accommodate the requirement for total schema visibility.
- A semantic knowledge leak may occur depending on the integration method used. If multiple LDBS schemas are integrated pair-wise rather than simultaneously, incomplete semantic knowledge may be used for later integration steps, thus propagating error.

- The schema integration process itself is time intensive and subject to error.
- A global schema integration is a static solution and inappropriate for dynamic schemas because the entire integration will need to be redone to accommodate changes to LDBSs. As the quantity and dynamism of LDBSs increases, the need for costly re-integration increases, therefore this approach scales poorly.

A variant of global schema integration, *partial schema unification/combination* involves integrating a subset of LDBS objects into a global schema. This approach has the same disadvantages as global schema integration, though on a smaller scale.

2.1.2.2 Federated Database Systems (FDBS)

The federated approach is a compromise between no and total integration. It permits more autonomy and flexibility than global schema integration, but has the following drawbacks: partial integration is done on demand with little automation; and database administrators have to integrate manually—a costly, repetitive process [1]. According to Kim [7], the database research community considers the FDBS to be the most viable and general MDBS strategy. The aim of the federated approach is to relieve the requirement for static global schema integration, which results in more association autonomy and decentralized control with less complete integration.

An FDBS architecture is typically comprised of the following components [1]:

- *Local Schema*—conceptual schema of the LDBS described in its own data model.
- *Component Schema*—local schema translated into the common data model (CDM) of the FDBS to resolve data model heterogeneity. Each LDBS stores mappings from its data model to the CDM.

- *Transforming Processor*—uses mappings between local and CDM objects to translate commands from FDBS to local format and data from local to CDM format.
- *Export Schema*—each LDBS specifies objects sharable to members of the FDBS.
- *Filtering Processor*—limits operations submitted against LDBS schemas based on access controls specified in the export schemas.
- *Federated Schema*—static integrated schema or dynamic user view of multiple export schemas.
- *Constructing Processor*—decomposes queries from the federated schema to local schemas and merges results.
- *External Schema*—another layer of abstraction that allows for classes of users and applications by specifying additional access control information and integrity constraints.
- *Data Dictionary*—contains the export, federated and external schemas; mappings between the schemas; metadata useful for optimization; and necessary data transformation functions.

The loosely coupled federated approach involves highly autonomous, read-only LDBSs. Each user maintains and creates the federation schema, and each user must know what data is available, where the data resides as well as its structure in the export schemas in order to create views.

Advantages include:

- Different classes of users can map different semantics to the same set of export schema objects via dynamic attributes.

- A loosely coupled FDBS can cope with dynamic LDBSs schemas better than a tightly coupled FDBS, because it's easier to construct a new view than to recreate a global schema.

Disadvantages include:

- Detecting changes to remote schemas can be difficult due to the volume of broadcast messages generated by triggers.
- Independent users maintain their own views, so there's potential for duplicate views.
- The potential for large numbers of export schemas results in increased complexity and attendant difficulty in understanding.
- Multiple semantic mappings make view updating problematic.

The intent of the tightly coupled federated approach is to provide location, replication and distribution transparency across one or more federated schemas. A single federated schema formed across all export schemas is essentially global schema integration. Federation administrators create and maintain federated schemas and access to export schemas. Multiple federation schemas provide flexibility based on role or frame of reference, but are difficult to maintain due to increased complexity and the possibility of semantic inconsistencies caused by enforcing multiple constraints in export schemas. Disadvantages of this approach include:

- Autonomy is violated because component schemas are visible to FDBS administrators during export schema negotiation/formation without data access.
- After creation, the federated schema is essentially static, and each federated schema must be completely redone when component/export schemas change.

- An external schema must be maintained which may differ from the federated schema, therefore additional translation and processing may be required.
- The data dictionary can become very complex with attendant problematic searching.

2.1.2.3 Multidatabase Language Approach [1]

The multidatabase language approach also favors autonomy, but users must perform schema integration across multiple databases themselves with support from the multidatabase language, which can access and manipulate a variety of databases. The primary advantage is the powerful database language itself in the hands of an expert user. The complete lack of distribution and location transparency is a major disadvantage. Users must find relevant data across multiple databases, understand each schema, detect and resolve semantic conflicts, and perform view integration. Another drawback is the lack of support for reuse--the integration must be performed every time.

2.1.2.4 Implications

There are two important issues that must be addressed by the various approaches: first, who must perform the integration and, second, what level of flexibility and autonomy are preserved. In the global schema approach, the DBA or another expert integrates the local schemas into an inflexible global schema. Any change to a local schema necessitates re-integration. The federated approach has the DBA performing the integration and achieves a moderate level of flexibility. The user performs the integration in the MDB language approach, which maximizes flexibility at the cost of total schema visibility.

2.2 *Research Issues*

Problems with the current state of data source heterogeneity resolution methodologies include:

- Most current methodologies are prototypical research projects that don't attempt to be full-scale automated systems, but can be used manually if the problem space is sufficiently small [8]. Additionally, they generally fail to provide algorithmic specifications of integration activities, but simply provide general guidelines.
- Termination of integration algorithms is typically left to the designer, rather than on quantifiable convergence.
- The behavioral specification of dynamic properties of schema objects is lacking.
- Schema mapping between local and integrated schemas isn't formally addressed in many methodologies.

2.2.1 Schema and Language Translation [1]

An MDBS must translate between local and global data models in order to resolve syntactic heterogeneity among LDBSs. The global model should be at least as rich as the data source models to minimize semantic loss caused by the inability to capture semantics in the global data model. The Entity Relationship model is predominant, but the object-oriented model is rising because it provides the ability to design and implement in the same model. The common access language used in an MDBS must be a subset of the local languages if a local feature isn't available in all LDBS. Data model transparency helps hide differences between query languages and data formats inherent to the diverse data sources.

2.2.2 Schema Integration

There are various reasons for the differences between DBMSs requiring integration[1]. Different organizations design and implement DBMSs to address their own unique requirements without knowledge or regard for future integration efforts that may occur. Each organization has different requirements and spends its development dollars on satisfying its needs. Technology changes over time, particularly information technology, and DBMSs designed at different times will have different technologies available to the designer.

Even if the same data model is used across LDBSs, schematic and semantic heterogeneity results from the various ways the world can be represented. The more powerful the data model, the more differences can occur. Different data models provide different structural primitives, and if the semantics are the same, structural or schematic resolution is easier, but if the semantics diverge, then resolution is problematic.

There are two types of naming conflicts: synonyms and homonyms. Synonyms have different names but the same semantics, and can be resolved via aliases and global schema constructs. Synonyms can only be detected via external specification.[8]. Homonyms are named identically but have different semantics, and can be detected via a thesaurus or semantic dictionary.

The following structural conflicts can occur between schemas:[8]

- Type conflicts—the same concept is represented by different modeling constructs in different schemas.

- Dependency conflicts—a group of concepts is related differently in different schemas, e.g. 1:1 vs *m:n* in a marriage relationship.
- Key conflicts—Different keys are assigned to the same concept in different schemas, e.g. VIN number vs license number to indicate an automobile.
- Behavioral conflicts—different insertion and deletion policies in different databases for the same entity, e.g. existence dependence versus identification dependence.
- Missing or conflicting data—semantically the same, yet have different/missing attribute value(s).

Semantic Heterogeneity—difference in meaning, interpretation or intended use of the same or related data caused by geographical and organizational aspects.

2.2.3 MDBMS Consistency and Dependencies [1]

The application-specific nature of interconnections between diverse information systems makes their integration a laborious, manual process. Automating integration is highly desirable, but difficult because of the need to specify and enforce consistency of interrelated data across DBMSs. Tentative efforts include active databases and interdependent data that capture structural dependencies via data dependency descriptors and maintain consistency via temporal, state components and procedures. Traditionally, MDBSs are read-only because updates cause concurrency control, logging and security problems.

2.3 Schema Integration [1:120]

Schema integration is key to both the global schema and federated database approaches. The input to the schema integration process is schemas representing the semantics of the LDBSs, and the output is an integrated schema in a common data model that captures the semantics with schematic heterogeneity resolved. Schema integration differs from view integration as follows:

- View integration is a top-down database design technique typically used to design new schemas, while schema integration is a bottom-up technique used to integrate existing databases.
- View integration typically involves a single data model, while schema integration must often resolve data model heterogeneity.
- View integration has more flexibility in semantic interpretation because it deals with abstractions, unlike the actual values dealt with in schema integration.

Schema integration is a labor-intensive process largely because most models can't capture the intended semantics completely and unambiguously. Schema integration, therefore, can't be completely automated, although tools can facilitate the process. An integrated schema must be updated as needed to reflect changes in LDBS structure, constraints or data values that result in changes to the schema.

2.3.1 Schema Integration Process

According to [20], schema integration methodologies are typically comprised of the following steps, which may need to be performed iteratively:

- *Schema Translation*—The LDBS schemas are translated into a common data model (CDM) that can completely capture the semantics represented in all LDBS schemas. Commands in the translated schema must be translatable into commands in the original LDBS schemas.

- *Schematic interschema relationship generation*—Generate a reliable set of relationships between database objects (entities, attributes and relationships) in the translated LDBS schemas. The relationships are identified and categorized by analyzing schematic properties such as integrity constraints, cardinalities and domains.

- *Integrated schema generation*—Generate an integrated schema from the interschema relationships identified in the previous step. This schema resolves the five categories of heterogeneity: domain definition, entity definition, data value, abstraction level, and schematic incompatibilities.

- *Schema mapping generation*—Store mappings between local and integrated schemas for use in query transformation.

Batini and Lenzerini [8] propose that the following activities comprise any integration methodology:

- *Pre-integration*—An integration policy that gives preference to certain schemas or portions of schemas is established based on schema analysis. A global integration strategy that determines designer interaction, the number of schemas to be integrated at a time, assertions and constraints among views is constructed.

- *Schema Comparison*—Schemas are analyzed to identify correspondence of concepts, detect conflicts, and discover interschema properties.

- *Conforming Schemas*—Resolve conflicts to enable schema merging. Automating conflict resolution is generally infeasible because designer and user interaction is needed to achieve compromises. In partially automated strategies, conflicts arising from fundamental inconsistencies are reported to users for resolution.

- *Merging and Restructuring*—Superimpose the schemas into an intermediate schema that can be analyzed and restructured, if needed, to improve its quality measured against the following criteria:
    - *Completeness and Correctness*—The integrated schema must accurately represent the union of the component schemas' domains.
    - *Minimality*—Concepts and relationships represented in multiple component schemas must be represented only once in the integrated schema to eliminate redundancies.
    - *Understandability*—The integrated schema must be represented as understandably as the data model allows.

2.3.2 Schema Integration Strategies[1]

Schema integration strategies can be categorized by the *abstraction level* at which integration is performed, by the *data model* used to represent the input schemas and by the *integration processing strategy*.

2.3.2.1 Abstraction Level[1]

The three abstraction levels where integration is attempted are *user view*, *conceptual schema* and *data* level. *View* integration is a top-down design technique that integrates user views typically represented in a common data model. Therefore the schema translation step is usually unnecessary. Strategies that integrate at the *conceptual schema* level can be divided into two classes:

- *Schema restructuring methodologies* apply schema-restructuring operators to generate an integrated schema from LDBS schemas possibly represented in heterogeneous data models. The resulting integration is static and the process must be repeated to reflect changes to the component databases.

- *View generation methodologies* integrate by developing views or defining queries against LDBSs. This is a more dynamic solution because changes to the component databases require regeneration of affected views rather than complete reintegration.

Data level methodologies rely on actual data values to perform integration and address the following problems:

- *Entity identification*—How to identify instances of the same real-world entity in different databases.
- *Attribute-value conflicts*—How to resolve different data values among instances of the same entity.

Data level strategies are inherently dynamic because changes to data values void previous integration.

2.3.2.2 Input Schema Data Model

The data model used to represent the input schemas determines the semantics that can be conveyed and must be resolved in the integration effort. The following four models are addressed:

- *Relational Models*—The advantages are the models' simplicity, the ubiquity of relational databases and the ability to formally address minimal redundancy. Data level approaches are attempted primarily on relational databases. The disadvantage is the model's limited expressive power and limited semantics that can be captured in the schemas[7,8]. Relational models also assume every relation name is unique; therefore naming conflicts and contradictory specifications aren't addressed. Some relational model methodologies use inclusion, exclusion and union functional dependencies to create disjoint subsets that enable expressiveness on par with the semantic models. [8].

- *Semantic Models*—These models use variants of the entity-relationship model, which are more semantically expressive than the relational model. Most of these strategies are user view and conceptual schema-level strategies because semantic models are frequently used to design views. The semantic models allow more freedom in naming and design perspectives [8].
- *Object-oriented Models*—These methodologies often integrate methods, complex attributes, and object hierarchies, as well as schemas. Most strategies are view and conceptual schema-level strategies.
- *Logic-based Models*—These models can formally capture semantics of relational databases and can capture more semantics than most semantic models. Relational schemas can be converted to logic models more readily than semantic models. Additionally, semantic integrity constraints can be defined for later use in query transformations.

2.3.3 Integration Processing Strategy[8]

Binary strategies integrate two schemas at a time and are more common because comparison and conforming activities are simpler. Strategies that integrate an additional component schema into intermediate results are *ladder* strategies, while *balanced* strategies are integrated symmetrically. The disadvantages of binary strategies are increased iterations of the integration process and the need for final analysis to determine and add missing global properties. The ladder strategy allows the integrator to prioritize the relative importance and order of integration of component schemas with an inherent preference toward the intermediate schema.

N-ary strategies integrate more than two schemas at a time, with *one-shot* strategies integrating all component schemas at once and *iterative* otherwise. Though the complexity

of integration at a step increases with this strategy, the number of steps is minimized and front-end semantic analysis can save further analysis and transformation of the final integration.

2.3.4 Interschema Relationship Identification (IRI)

It is logical to classify IRI efforts by abstraction level because the abstraction level determines the semantic knowledge available.

Conceptual Schema-based IRI approaches use a two-phase process: first identify relationships among objects, then classify the relationships. The first phase requires gleaning intended semantics to identify a set of potentially related objects. The second phase involves categorizing these relationships, which is a domain expert-intensive operation due to the limitations on the ability of various data models to completely capture true semantics. Relationships are identified from schematic constructs, e.g. entity classes, attributes, relationships, uniqueness properties, cardinality constraints, domains, integrity constraints, and set of allowable operations. All schematic constructs are candidates for relationship identification under one methodology or another. Due to the vast pool of input data, sophisticated dictionary/thesaurus mechanisms can greatly aid the IRI effort.

Having identified relationships, the next step is to classify the relationships, which varies according to the methodology. Example classification schemes include:

- Larson's [25] EQUALS, CONTAINS, CONTAINED-IN and OVERLAP.
- Ramesh and Ram's [26] degrees of similarity and dissimilarity. This scheme lends itself well to automation.

- Song's [27] WEAK, COMPATIBLE, EQUIVALENT, and MERGABLE semantic relations.

Data-based IRI approaches attempt to determine instances of entities in different databases that refer to the same real-world entity. The simplest approach operates under the assumption of a common key, with common key values identifying the same entity. A key equivalence problem results if a common key doesn't exist. Probabilistic techniques have been developed to determine the likelihood that two entities are the same through examination of key and sometimes all attribute values. Some methodologies combine schematic and data-level knowledge to determine attribute equivalence. Another methodology extends key equivalence to include instance-level functional dependencies. Finally, Ramesh and Ram [26] propose a technique to determine relationships based on integrity constraints. This data-level technique attempts to identify entity-class relationships rather than entity-instance relationships via data-value examination.

2.3.5 Integrated Schema Generation (ISG)

Integrated schema generation efforts are best classified by the model-level classification because the data model, semantics and heterogeneity of component databases are the main drivers of the ISG process.

The primary technique used in *semantic model approaches* is the creation of generalization/specialization relationships in the integrated schema. Larson's [25] approach assumes that if an object pair can integrate their identifying attributes, then the objects can be integrated. Larson's categories of relationships and guidelines for transforming related objects into the integrated schema are widely used in the various

semantic model techniques. El-Masri and Navathe [28] present techniques for integrating Larson's possible entity class and relationship pairs into an Entity-Category Relationship model. These rules implicitly require naming conflict resolution. Naming conflicts appear as unrelated objects with the same name, requiring renaming of one object in the integrated schema, or as equivalent objects with different names, requiring a single name to be chosen for inclusion in the integrated schema. Structural conflicts are caused by different data model constructs or like constructs with different properties. Most methodologies address naming and structural conflicts.

Object-oriented approaches address all issues resolved by those based on semantic models, as well as integrating class hierarchies and methods. Class hierarchies represent classes participating in generalization/specialization relationships, which may be recursive in nature. The primary focus of Sull and Kashyap's [5] methodology is *integrating class hierarchies*. Their methodology is self-organizing with propagation of local schema updates to the integrated schema. It also demonstrates mapping from relational schemas and rule bases to object-oriented schemas. *Method integration* involves defining new methods for an integrated view and integrating preexisting methods into the integrated view while resolving name and parameter conflicts.

Logic-based approaches are in their infancy, but are valuable due to the powerful semantics that can be conveyed by logic systems as well as their formal nature. In a rule-based approach, local schemas are expressed in first-order logic as extensional databases (EDBs). The integrated schema is expressed as rules applicable to the EDBs, which is a set

of intentional database (IDB) relations. Query processing is simplified because most SQL-based languages are readily translated to logic-based queries.

2.3.6 Schema Mapping Generation

Mappings generated during the schema translation process translate between integrated schema objects and local schema objects. This mapping may be stored as a dictionary at each database as well as in a global catalog.

2.3.7 Automating Schema Integration

Schema integration can't be completely automated due to incomplete semantic representation caused by model limitations and differences in integration due to the intended use of the integrated schema. However, much of the tedium can be automated to lighten the burden on designers and domain experts. Several toolkits have been devised to this end. DeSouza [29] developed an expert system IRI aid for schemas defined in Abstract Conceptual Schema (ACS). A set of resemblance functions use name and structure to estimate resemblance between constructs. Associated weighting factors indicate the relative importance of each criterion, which allows the user to tune the process. Unfortunately, DeSouza's methodology can only be used on ACS schemas, and it doesn't address the integrated schema generation step. One of the most promising toolkits, presented by Ramesh and Ram [26], measures similarity and dissimilarity between entity classes, attributes and relationships. A four-level blackboard architecture is used to support the schema integration process.

The problem with current IRI automation techniques is that they use schematic information as their sole knowledge source for determining relationships, which can lead to incorrect inferences. Multiple knowledge sources, including integrity constraints as well as sources produced from data mining and information retrieval techniques, can likely improve this process. Though ISG techniques to resolve inter-schema structural differences are relatively mature, other areas need improvement. One pressing unresolved issue is the efficient management of the schema evolution required in dynamic environments.

2.4 Sull and Kashyap's Schema Integration Methodology [12]

The schema translation algorithm proposed by Sull and Kashyap was selected as the initial algorithm for this research. The goal of their methodology is to achieve "a seamless sequence of schema translation and integration processes, which can prevent ambiguities in update propagation from component schemas to integrated schemas." The essential property of this methodology is *self-organizability*, which claims to sustain the integrated schema's semantic integrity regardless of updates to local schemas. The input schema is assumed to be "in Boyce-Codd Normal Form" and domain constraints must be available for comparison.

Sull and Kashyap make the following claims with regard to their strategy. An injective mapping from relational to OO schemas can be defined so that local schema updates can be unambiguously propagated to the equivalent export schema. When a

relation is mapped to an object, it can be classified based on interconnectivity of zero, single, or multiple connections. These three cases of interconnectivity and potential generalization hierarchies are handled by the algorithm.

The *information content* of a schema defines the schema's legal states, and two schemas are *content-equivalent* if there is an "invertible, attribute-preserving mapping between their possible instantiations." Further, "a schema transformation T is *content-preserving* if for every schema S there is an equivalence mapping to T(S)." Sull and Kashyap's strategies are content preserving from relational to OO databases. Since the translated schemas are used to create the integrated schema, an accurate translation is essential. Sull and Kashyap's algorithm claims to meet this requirement.

2.4.1 Schema Translation Algorithm Description

The input to Sull and Kashyap's schema translation algorithm in Figure 1 is an arbitrary subset of relations from a given schema. The output is a set of object classes and three sets of links: generalization, aggregation and interaction. Generalization links capture "is a" relationships, aggregation links depict "composed of" relationships and interaction links capture any relationships not addressed by generalization or aggregation. Links are generally created between two object classes, but in many cases links are created between object classes and relations. This occurs because Sull and Kashyap defer object class creation until the schema integration phase except in the simplest case of relations without imported keys. The three link sets are later converted into restructured hierarchies during the subsequent schema integration phase, where necessary object classes are created to

resolve links between object classes and relations. Deferring object creation and link resolution until integration helps the methodology incorporate missing relations that may be provided by other translated schemas.

A simplified translation of Sull and Kashyap's schema translation algorithm follows:

- Step 1: Create corresponding object classes for relations with single-attribute primary keys.
- Step 2: Create a set of generalization links between object classes by identifying subset relationships between the instance values of the primary key in tuples of the parent relations.

Relational to Object Oriented Schema Translation Algorithm:
Let $R(A_1, A_2,...A_n)$ be a relation where A_i's are attributes associated with relation R. Let PK be a primary key consisting of a set of one or more attributes. Let FK be a foreign key consisting of a set of one or more attributes. Let $t[A_i]$ refer to the value for attribute A_i in tuple t. 1. For each relation R with a single-attribute PK: Create an object class O_r. 2. For R_1, R_2 with PK_1 and PK_2, if $\{t[PK_1] \supseteq t[PK_2]\} \rightarrow R_2$ is a subclass of R_1 (vice versa). Create a G link between O_{r1} and O_{r2}. 3. For each relation R with a single FK, Identify object classes $\{O_i\}$ where each class has PK whose domain is equivalent to that of the FK of relation R. If O_i is itself, create unary I link on O_r. If O_i is another object class, create an I link between O_i and O_r. 4. For each relation R with multiple attributes and multiple FK's, Identify object classes $\{O_i\}$ where each object class has a PK with equivalent domain of each attribute of relation R. Create an A link between O_i and R where $i = 1...n$. 5. For each relation R with a composite-attribute PK, Identify object classes $\{O_i\}$ where the PK of each class has equivalent domain to each PK of relation R. Create an n-ary I link among R and O_i where $i = 1...n$.

Figure 1: Sull & Kashyap's Schema Translation Methodology

- Step 3: Create a set of interaction links between relations with single foreign keys and objects by identifying domain equivalence between a parent relation's primary key and a foreign key.
- Step 4: Create a set of aggregation links between relations with multiple foreign keys and object classes whose parent relation's primary key is domain equivalent with the attributes of the multiple-foreign-keyed relation
- Step 5: Create a set of interaction links between composite-primary-keyed relations and object classes with primary keys that are domain equivalent with component attributes of the composite primary key.

Thus, the output of the translation process is a set of object classes and sets of generalization, interaction and aggregation links that are later used to develop an integrated schema. During schema integration, the links are placed in a hierarchy, which is restructured to create missing superclasses and remove redundant links.

III. Methodology

The methodology steps presented define a process for developing an automated application for translating heterogeneous data sources into a common object model to facilitate integration. The schema translation algorithm proposed by Sull and Kashyap and discussed in Chapter II is applied to a base-case relational schema. The results of this application are analyzed and a revised schema translation algorithm is proposed by this research. In Chapter IV, the revised algorithm is implemented and applied to a validation case relational schema to verify its efficacy.

3.1 Methodology Overview

The process for generating a global integrated schema in an object model is depicted in Figures 2 and 3. Prior to integration, the diverse schemas are translated into common data models to facilitate integration. The object model was chosen because of its many advantages [13,14] and to facilitate interoperability between this research and related efforts at AFIT [12, 15, 16, and 17]. The proposed methodology addresses the schema translation activity on the left of the dotted line in Figure 3, which provides the input to the remainder of the schema integration effort on the right [16]. The contribution of this effort is automating the translation process where possible, as well as identifying and describing areas where complete automation is not yet feasible.

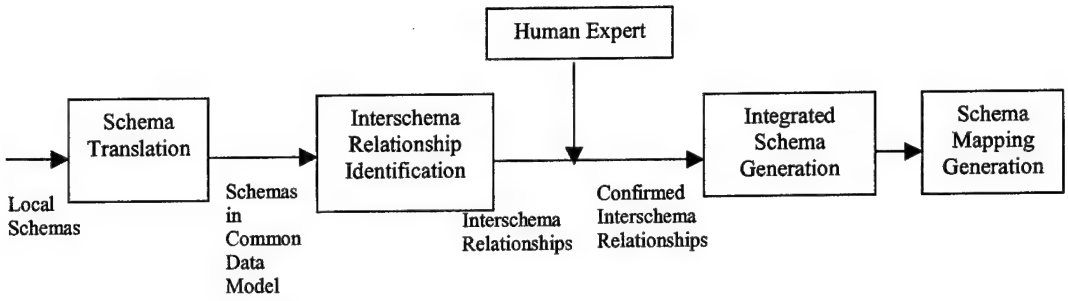

Figure 2: Steps in a Schema Integration Methodology [1]

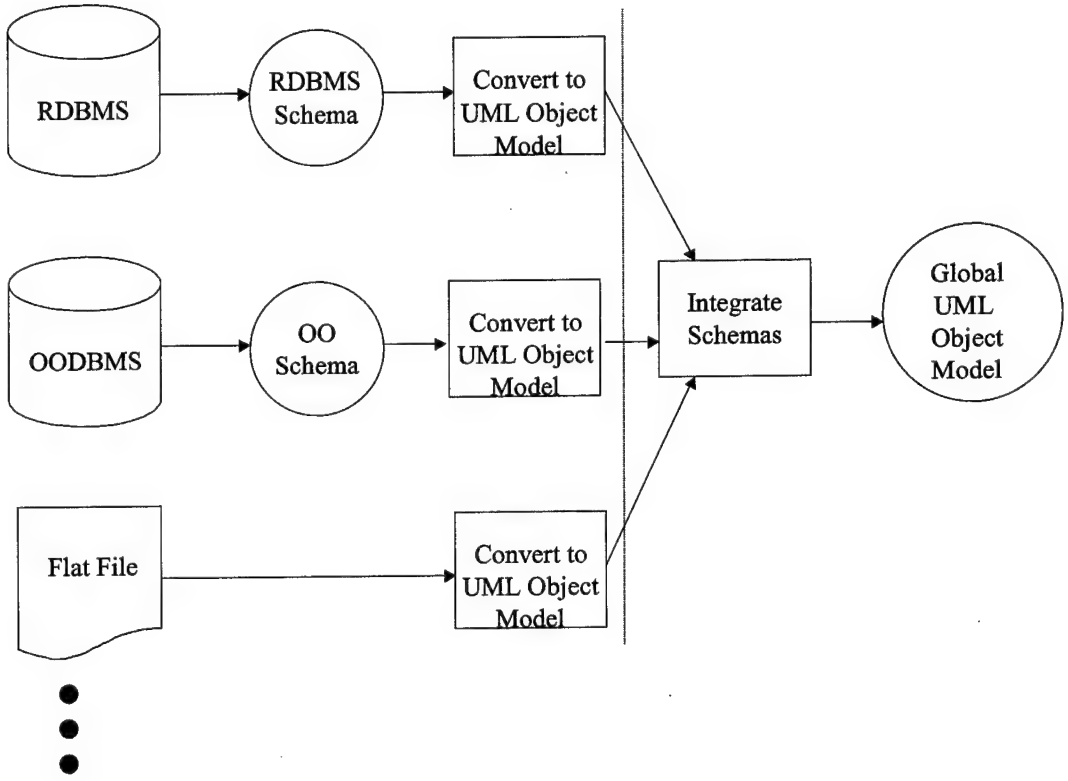

Figure 3: Generating a Global Schema in an Object Model

The primary objectives of the proposed methodology are the following:

1. *Translate an RDBMS schema into a UML Object Model.* The research will begin with Sull and Kashyap's schema translation methodology [12], with extensions as needed.
2. *Update the UML schema to reflect incremental changes to the RDBMS schema.* This entails detecting changes to the relational schema and propagating updates to the UML schema.
3. *Determine the feasibility of automating the update process.* Some aspects of translated schema maintenance are easily automated, some aspects may be impossible to automate due to required human intervention, and still other aspects lie somewhere in between. We provide a characterization of the feasibility of automating the various aspects.
4. *Address other data source types.* The second data source type considered is an XML file.
5. *Produce interoperable output.* Initially, the translated schema resides in a preliminary object-oriented representation (POOR) within the translation application, which will act as a metadata repository against which the translation application can work. In order to achieve interoperability with related AFIT research efforts, this research translates from the POOR into syntax that can be imported into the AFIT Wide-Spectrum Object Modeling Environment (AWSOME) metamodel [17].

3.2 Schema Translation from Relational to Object Model

The relational schema translation task flow that was devised to support the preceding objectives is as follows:

1. Design the schema translation application.
    - Select an initial schema translation algorithm.
    - Develop an internal representation for the relational schema.
    - Develop an intermediate representation for the translated object–oriented schema.
2. Analyze the initial translation application.
    - Select or develop a base-case relational schema.
    - Analyze and apply the initial translation algorithm against the base-case relational schema to verify its utility to this research effort.
3. Improve the schema translation application, if needed.
    - Modify the initial schema translation algorithm as needed to devise an improved algorithm that maximizes utility for this research.
    - Apply the improved schema translation algorithm to the base-case relational schema to verify its utility.
4. Validate the schema translation application.
    - Select a validation-case relational schema.
    - Apply the improved algorithm against the validation-case schema.
5. Produce interoperable output.
    - Map the intermediate object-oriented schema representation to AWSOME syntax.

6. Dynamically maintain the schema translation.
    - Evaluate temporal and event-driven approaches to detecting changes to the relational schema and regenerating a translated schema.
    - Implement a schema change detection and translation regeneration scheme.
7. Formulate conclusions.
    - Document results and propose a general schema translation methodology.

3.2.1 Base-case Relational Schema

The schoolhouse example depicted in Figures 4 and 5 was selected because it is complex enough to exercise the selected schema translation algorithm, yet is simple enough to facilitate the verification of results via inspection. It began as the object oriented schema

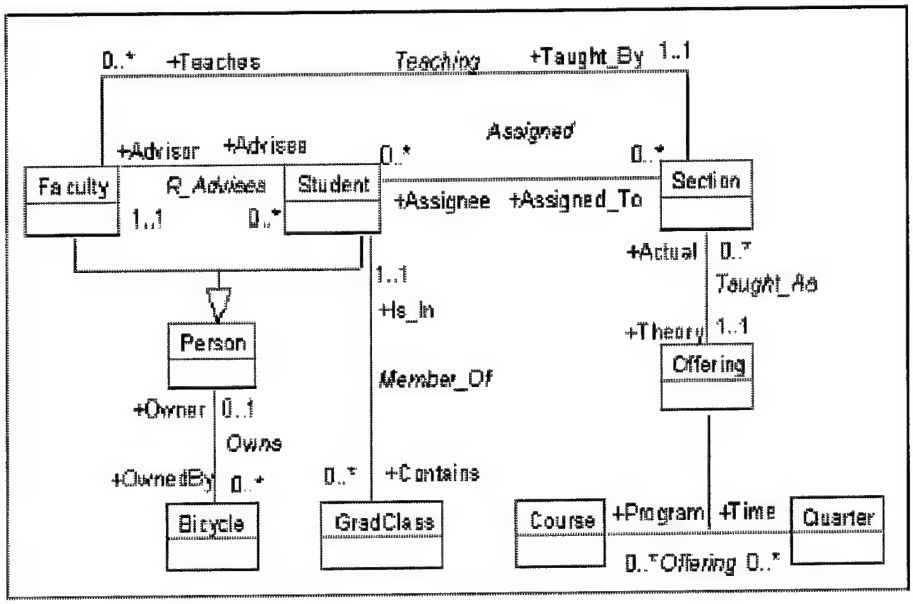

Figure 4: Schoolhouse Object-Oriented Schema

shown in Figure 4, which was translated into the equivalent relational schema seen in Figure 5 by a related AFIT research effort [12]. The Data Definition Language (DDL) produced by [12] that created the schema is included as Appendix A. The Person relation duplicates information stored in the Faculty and Student relations, which would not be the case in a typical relational schema. The methodology must address this case to preclude aberrant results from this atypical schema, possibly by removing the duplicative relation from the schema.

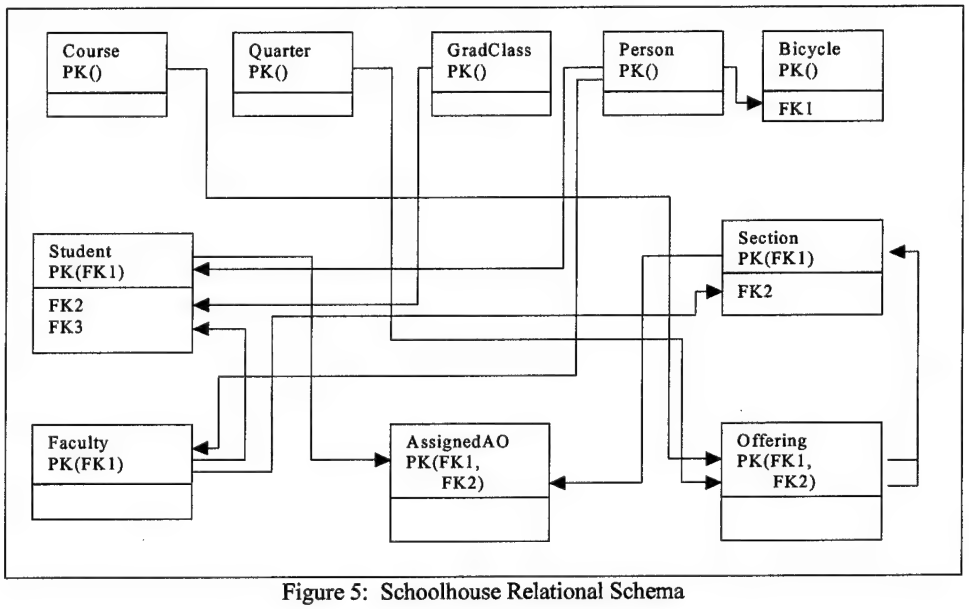

Figure 5: Schoolhouse Relational Schema

3.2.2 Internal Representation of the Relational Schema

The schema-level information in Figure 6 must be captured from the relational database to support schema translation. This data is directly retrievable from the DBMS data dictionary or indirectly from tools that access the data dictionary to service requests for metadata. For example, USER_TABLES contains metadata describing all tables, and USER_TAB_COLUMNS contains metadata describing all columns in those tables.

```
For each Schema
        Tables that comprise schema
        For each Relation
                Table name
                Primary key name and composition
                Foreign key name(s) and composition
                Column names
                For each Primary Key
                        Primary key name
                        Column name(s)
                For each Foreign Key
                        Foreign key name
                        Column name(s)
                        Update and Delete rules
                        Deferability of rules until commit
                        Primary key table
                        Primary key name
                        Primary key column name(s)
                For each Column
                        Column name
                        Column size
                        Data type
                        Nullability
                        Default value
```

Figure 6: Schema Level Information

Instance-level information required is depicted in Figure 7. This data is accessible by examining instances of relations in a database, though some may be automatically generated by the DBMS and stored in the data dictionary for use in optimization.

```
For each Relation
    Cardinality of tuples
    For each Primary or Foreign Key
        Cardinality of unique instances in index columns
    For each Column
        Presence or absence of values
        Range values of instances
        Uniqueness of instances
```

Figure 7: Instance Level Information

3.2.3 Preliminary Object-oriented Representation for the Translated Schema

The translated schema must capture the information in Figure 8, which is analogous to the schema and instance level information shown in the previous section.

```
For each Schema
        Object Classes that comprise schema
        For each Object Class
                Class name
                Attribute names and composition
                Links with other classes
                For each Attribute
                        Name
                        Size
                        Data type
                        Nullability
                        Initialized value
                For each Link
                        Role name
                        Destination object class name
                        Type (aggregation, generalization, interaction)
```

Figure 8: POOR Information Required

3.2.4 Results of Sull and Kashyap's Schema Translation Algorithm

Sull and Kashyap's translation algorithm, described in Section 2.6, produces the object classes and links in Figure 9 when applied to the schoolhouse example.

- Step one creates object classes for all relations with single-attribute primary keys, which produces the set of object classes.
- Step two identifies subset relationships among the relations' primary key instance values and creates the set of generalization links.
- Step three tests for domain equivalence between the foreign key of relations with a single-foreign key and every relation's primary key, which generates members of the interaction link set depicted above the dashed line.

- Step four tests for domain equivalence between foreign keys of relations with multiple foreign keys and every relation's primary key, which produces the set of aggregation links.
- Finally, step five identifies domain equivalence between components of composite primary keys and the primary keys of other relations, which generates the remaining members of the interaction link set.

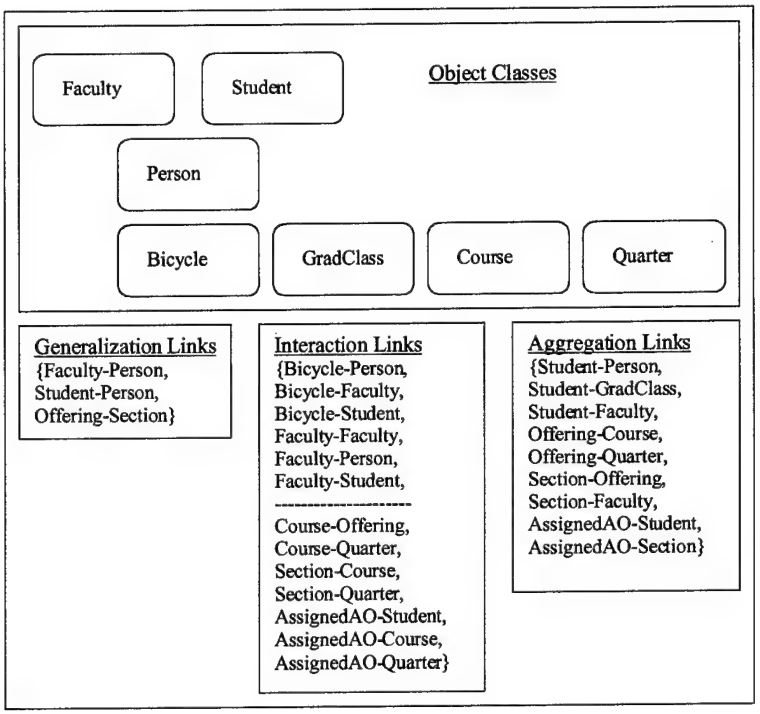

Figure 9: Results of Algorithm Applied to Schoolhouse Example

3.2.5 Analysis of Sull and Kashyap's Schema Translation Algorithm

Several issues arose during application of the translation algorithm that require resolution in order to make the algorithm suitable for this research. These issues range from ambiguities like domain equivalence determination to omissions, e.g. inter-entity cardinality determination and role name assignment.

3.2.5.1 Domain Equivalence

The first issue with the algorithm that wasn't rigorously treated in the reference, and therefore requires resolution, is the concept of *domain equivalence.* If domain equivalence is treated as simple syntactic equivalence based on data type, erroneous associations are often generated as seen in the results above. For example, if varchar2(9) is used for both bicycles' serial numbers and persons' social security numbers, the domains are treated as equivalent despite significant semantic disparity. Range value examination at the instance-level can resolve some cases, but doesn't resolve the serial number versus social security number problem and provides only a transient solution at best. Domain constraints beyond simple data-typing are generally addressed at the application level rather than by the DBMS, and a limited number of data types can represent an infinite set of domains, so syntactic equivalence alone is insufficient.

Going beyond a strictly syntactic interpretation toward a more robust concept of semantic equivalence generally increases the amount of user intervention required, which precludes the research goal of automating schema translation. A middle ground that can be automated involves tracing migrated keys back to their parent relation, which guarantees designer-intended semantic equivalence between the attributes in question. As seen in Figure 10, *PK1* and *FK2* are semantically equivalent due to designed key migration, as are *PK3* and *FK4*. However, even if *PK1* and *PK3*, and therefore *FK2* and *FK4*, are semantically equivalent, key migration alone can't establish this fact. Therefore, tracing migrated keys improves the accuracy of link generation, except in step five where domain equivalence determinations between primary keys are required.

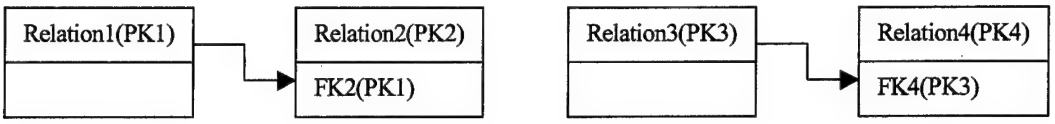

Figure 10: Attribute Equivalence

3.2.5.2 Arbitrary Subsets of Schemas

A closely related issue with the translation algorithm is the provision for accepting an arbitrary set of a relational schema as input to the translation process. This often prevents tracing migrated keys back to their parent relation, since the parent relation may not be included in the subset of relations in consideration. If an entire schema was required as input rather than an arbitrary subset, this problem would be solved. However, if a *complete subset* of a schema can be defined as a subset of relations that includes parent relations for all migrated keys of relations in the subset, then relaxing an *entire schema* requirement to a *complete subset* requirement will resolve the problem in a less drastic fashion. Further, if an entire schema is translated, and relations not participating in key migration are later added to the schema, then a translation of a complete subset of the schema has been accomplished via the initial translation. This fact allows testing of the complete subset provision of this methodology to be implicitly included while translating an entire schema.

3.2.5.3 Delayed Object Creation

Another issue concerns delaying object creation until the schema integration phase for relations that don't have a single-attribute primary key. Integration is outside the scope of this research effort. This often causes link creation between object classes and relations, and others between relations that don't become objects until a subsequent integration

phase, which results in a hybrid schema. These object-to-relation and relation-to-relation links allow the methodology to flexibly piece together subsets of schemas during integration. Missing objects or "holes" caused by schema subsetting in one translated schema may be present in another schema, which can fill in the holes during integration. One potential solution for this issue would be to treat the translation process as an integration of a single schema and proceed with the integration phase of Sull and Kashyap's methodology. However, this research diverges on this point and creates object classes when the need for each class is identified during link creation. Further, links are stored with their associated object classes rather than as separate sets.

3.2.5.4 Extraneous Link Generation

As seen in Figure 11, the algorithm generates extraneous and duplicative links, causing redundancies that must be eliminated during integration. An inter-entity relationship may be represented as both interaction and aggregation, e.g. Faculty-Student, or as both aggregation and generalization, e.g. Section-Offering. In this research, redundant links are removed during the translation phase rather than postponing restructuring until integration. This improves the interoperability of the research results because redundant links increase the complexity of any subsequent integration effort, and removal of extraneous links makes the translated schema a far better candidate for integration. The POOR retains all links generated by the translation algorithm regardless of duplication, but duplicative links aren't generated in the AWSOME output. Retention of redundant links in POOR captures design decisions that may be useful during integration tasks outside the scope of this research, e.g. inter-entity relationship identification.

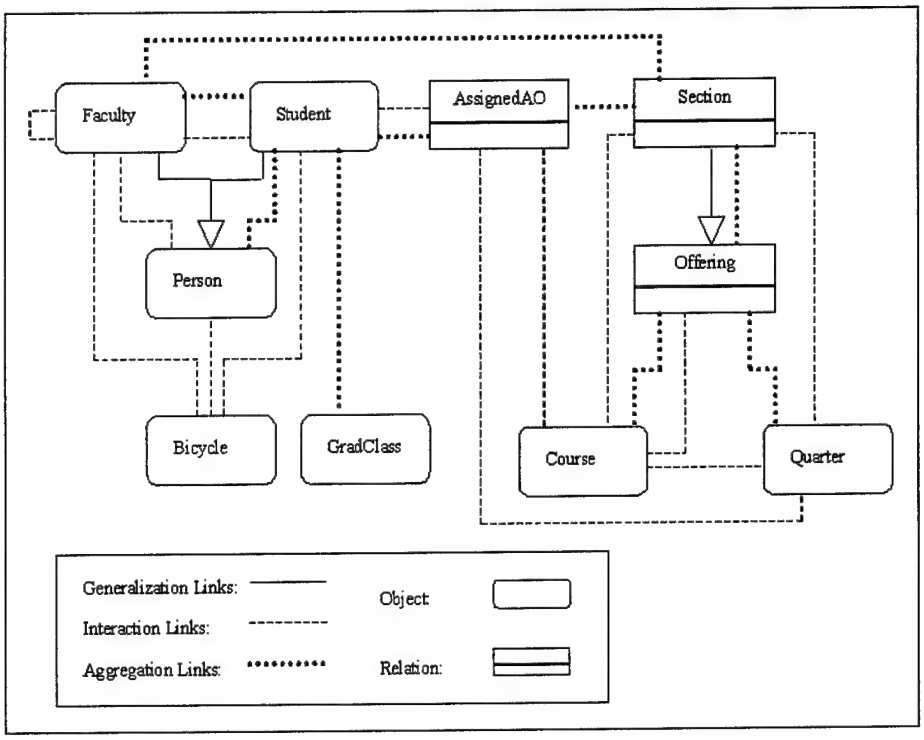

Figure 11: Resulting Hybrid Schema

3.2.5.5 Tracing Migrated Keys

Step four of Sull and Kashyap's algorithm requires a domain equivalence determination between each attribute of multiple-foreign-keyed relations and the primary keys of every object class. This has the effect of identifying possible parent relations for foreign keys in multiple-foreign-keyed relations. Based on this understanding and the aforementioned complete-subset provision, this research traces foreign keys directly to their parent relation.

3.2.5.6 Inter-entity Relationship Cardinalities

Object-oriented representations generally require cardinalities to be assigned to relationships among object classes. This isn't explicitly addressed by Sull and Kashyap's

translation algorithm and is problematic when translating from a relational schema. The difficulty in qualifying cardinalities is partially caused by there being no single source for the information necessary to accurately determine and thus assign cardinalities.

Systematic tracking and representation of key migration is the primary method used to establish and enforce relationships in a relational schema, as can be seen in Figure 12. If the primary key of a parent relation is used as all or part of the primary key for a child relation, then an *identifying* and *existence-dependent* relationship is formed. If one or more attributes of the migrated key are not included in the child's primary key, then a *non-identifying* relationship is formed. If the migrated key can be null in a child instance participating in a non-identifying relationship, the relationship is optional because the child instance is existence-independent. Conversely, a mandatory non-identifying relationship involves an existence-dependent child that requires a value in the migrated key field.

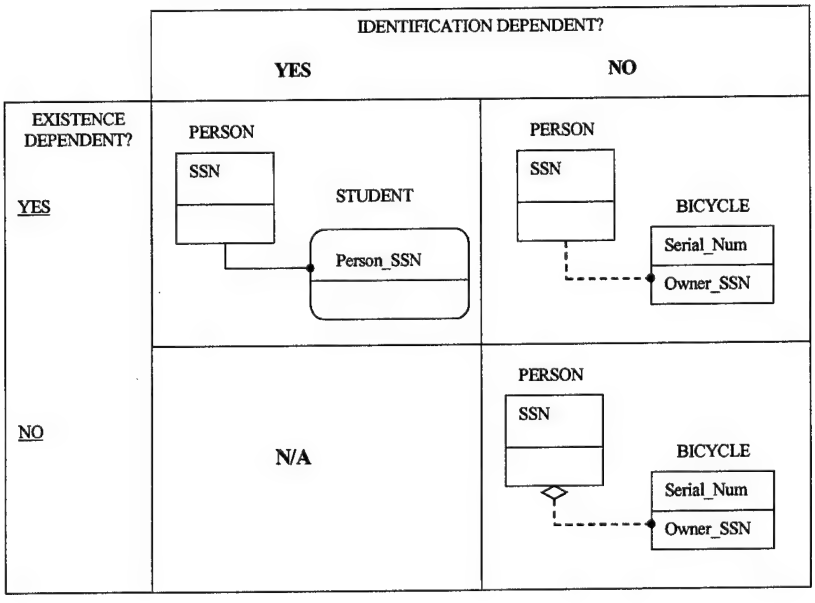

Figure 12: Relationship Dependencies [4]

The following levels of examination can be used to establish and refine relationship cardinalities: *schema* and *instance*. The schema level captures semantics specified for enforcement by the designer, while the instance level reflects the current state of the data. Some instance-level information is enforced by the DBMS, while other instance-level information is decided by random occurrence. Inspections at both the schema and instance levels may be required to make a cardinality determination. Concrete, schema-level evaluation is preferable to relying on relatively ephemeral, instance level information, but instance level examinations, even to the point of "data mining," may generate supplemental clues that aid translation accuracy. The following dimensions can characterize the cardinality of inter-entity relationships:

Schema Level

- Column nihility
- *Uniqueness* enforced on column instances
- Update and delete rules for imported keys
- *Triggers* that supplement referential integrity
- Key composition

Instance Level

- *Cardinality* of unique instances in index columns
- *Cardinality* of tuples in relations
- Presence, absence and range of values in column instances
- *Uniqueness* of column instances

In identifying relationships, if the migrated key comprises the entire primary key of the child as seen in Fig 13(a), then the cardinality of the relationship is exactly one parent for every child, and duplicate child entities aren't permitted. If a child instance is required for every parent instance, then the relationship cardinality is exactly one child for every parent, but without update rules each parent can have zero or one child. However, if the child's key has additional columns, as seen in Fig 13(b), there can be zero, one or more children for each parent unless an update rule restricts the relationship to one or more children for each parent. When any additional fields are another migrated key (or keys) and no non-key attributes exist, an intersection entity is formed. An associative entity differs from an intersection entity by having non-key attributes as well. Intersection and associative entities generally indicate *m* to *n* relationships.

Typically a DBMS forces the designer to declare the primary key NOT_NULL, but designers can also use this capability for non-primary key fields. Nihility of the migrated key establishes the cardinality's lower bound for the parent in non-identifying relationships. If the migrated key can be null, the cardinality is zero or one; otherwise, exactly one parent exists for each child entity. If the designer doesn't enforce nihility, instance-level information can be used for cardinality assignment. Specifically, if the cardinality of the child's foreign key is less than the cardinality of the parent's primary key, then the cardinality is zero or one parent to each child entity.

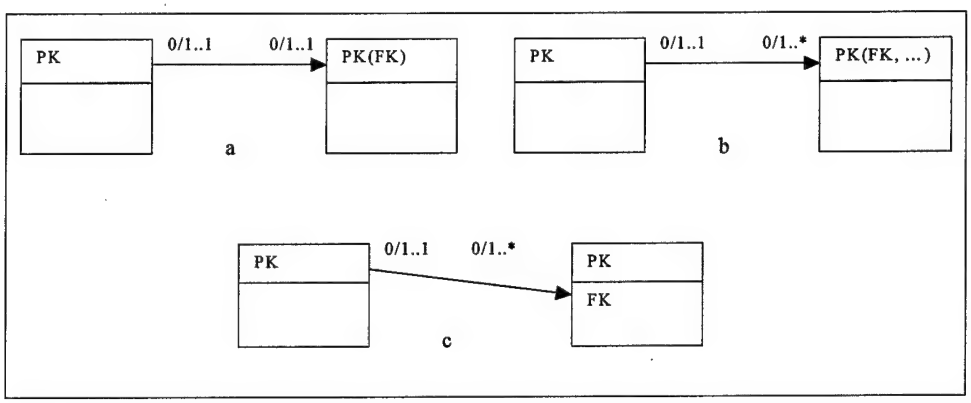

Figure 13: Relationship Cardinalities

3.2.5.7 Relationship or Role Names

Sull and Kashyap's algorithm doesn't address relationship naming, yet a naming convention that maximizes semantic utility must be devised. A combination of the parent entity's name, the parent's primary key, the child entity's name and the child's foreign key lends itself well to automation and preserves available semantics well. For example, EMPLOYEE.EmployeeNumber-SPOUSE.SponsorID isn't the most succinct or expressive role name, but it captures semantics better than a strictly numerical name. Using entity names alone is insufficient to differentiate relationships because multiple relationships can exist between the same entities. If relationships are treated as bi-directional, the role name can be reversed to enhance understandability in the opposite direction.

3.2.6 Revised Schema Translation Algorithm

To incorporate lessons learned from applying Sull and Kashyap's translation algorithm against the schoolhouse example, the revised algorithm is depicted in Figure 14. Modifications to the initial algorithm are italicized. Step 1 specifies the *complete subset*

requirement. Link (association) cardinality determination and previously non-addressed object creations are addressed by Step 3 through Step 6.

1. *Select a complete subset of relations from a relational schema for translation.*
2. Create corresponding object classes for each relation with a single-attribute primary key.
3. Create generalization links.
   For all relations R1 and R2
   If the instance values of R1's PK are a subset of the instance values of R2's PK, then R2 is a subclass of R1
   If associated object classes for R1 and R2 don't exist, create them
   Create a generalization link between O1 and O2
   Determine and assign link name and cardinality:
   *Source cardinality is **zero-to-one** and destination cardinality is **zero-to-n**.*
4. Create interaction links between single-foreign-keyed relations and their parent.
   For all relations
   If R1 is single-FK'd, then R2 is the parent relation
   If associated object classes for R1 and R2 don't exist, create them
   Create interaction links between R1 and R2
   Determine and assign link name and cardinality
   *If child is non-unique and the child's FK cardinality is less than parent's PK cardinality, then source cardinality is **zero-to-n** and destination cardinality is zero-to-one.*
   *If child is non-unique and the child's FK cardinality equals the parent's PK cardinality, then source cardinality is one-to-n and destination cardinality is **one-to-one**.*
   *If child is unique and the child's FK cardinality is less than the parent's PK cardinality, then both the source and destination cardinalities are **zero-to-one**.*
   *If child is unique and the child's FK cardinality equals the parent's PK cardinality, then both the source and destination cardinalities are **one-to-one**.*
5. Create aggregation links between multiple-foreign-keyed relations and their parents
   For all relations
   If R1 is multiple-FK'd, then R2..Rn are the parent relations
   If associated object classes for R1..Rn don't exist, create them
   Create aggregation links between R1 and R2…R1 and Rn
   Determine and assign link name and cardinality as in step 4.
6. Create interaction links between composite-primary-keyed relations and object classes with syntactically equivalent domains.
   For all relations
   If R1 is composite-PK'd, then R2...Rn have syntactically-equivalent PK's to column(s) in the PK of R1
   If associated object classes for R1..Rn don't exist, create them
   Create interaction links between R1 and R2…R1 and Rn
   Determine and assign link name and cardinality
7. *Generate AWSOME output--types, object classes, subclasses and associations*

Figure 14: Revised Schema Translation Algorithm

3.2.7 The AWSOME MetaModel [12]

Objective five of this research is to produce an interoperable representation of the translated schema. Therefore, the preliminary object-oriented representation described in Section 3.3.5 is translated into AFIT's AWSOME syntax, as seen in the sample depicted in Figure 15. AWSOME associations explicitly describe aggregation and interaction links identified during translation, e.g. Assigned, while generalizations are implicitly captured via "is *Class* with" syntax, e.g. Student.

```
package Schoolhouse is

type    zeroToN         is range 0..100000;
type    oneToN          is range 1..100000;
type    zeroTo1         is range 0..100000;
type    oneTo1          is range 1..100000;
type    integer         is range -1000000..100000;

type    Date            is array [1..9] of Char;

type    StringSet       is Set of String;
type    FacultySet      is Set of Faculty;
type    StudentSet      is Set of Student;
type    BicycleSet      is Set of Bicycle;

Class GradClass is
var program             : String;
var year                : Integer;
var graddate            : Date;
var designator          : String;
end Class;

Class Student is Person with
var gpa         : Integer;
end Class;

Association Assigned is
role    Assignee        : Student       is oneTo1;
role    AssignedTo      : Section       is zeroToN;
end Association;

end package;
```

Figure 15: Sample AWSOME Syntax

3.2.8 Approaches to Automating Schema Change Detection and Revised Translation Generation

Two approaches to schema change detection and revised translation generation are examined in this research—*event-driven* and *temporal*. In both cases, the initial application of the translation algorithm produces a baseline schema that is compared against subsequent executions of the algorithm to detect schematic changes. This allows a revised schema translation to be generated only if a change is detected, which precludes unnecessary re-integration.

The *event-driven* approach re-applies the translation algorithm upon execution of database commands that modify the schema, e.g. ALTER, CREATE or DROP TABLE. If the generalization hierarchy is to remain current, updates such as INSERT and DELETE must also trigger schema change detection, because added tuples may invalidate a previously identified generalization, and deletions could permit discovery of a new generalization. Additions and deletions can also change cardinality determinations that are based on instance-level information. The obvious benefit of the event-driven approach is that the algorithm isn't applied unless a schema change is possible, therefore a schema translation for a static database is maintained at no cost. Additionally, translation currency is maximized because changes immediately trigger an updated translation. Cost to achieve maximum translation currency is minimized because the algorithm is run the minimum number of times needed to guarantee availability of the most current translation. However, dynamic environments would cause frequent re-applications of the translation algorithm and consequently frequent revised translations would be forwarded to the integration

algorithm, potentially to the point of integrator saturation. In the worst case, the translation application would generate another translated schema as quickly as the application can cycle. The integrator could mitigate this by ignoring updates at its discretion, but the translator itself could saturate if the algorithm is triggered faster than it can be applied. If translation could be accomplished without cost, saturation wouldn't be a problem, but a cost-free algorithm is improbable, at best.

Since translation currency is primarily of interest to the integrator, it would be advantageous if currency could be tuned by the integrator. A *tunable-temporal* approach re-applies the translation algorithm at an integrator-specified interval to achieve the desired level of translation currency, which negates the possibility of integrator saturation seen in the event-driven approach. Translator saturation could still occur if the selected re-application interval is smaller than the time needed to complete each translation, but setting a re-application interval above this minimum threshold will effectively prevent saturation. In a quiescent database environment, the temporal approach will cause unneeded re-applications of the translation algorithm.

IV. Results and Analysis

This Chapter details the implementation and analysis of the methodology tasks described in the Chapter 3.

4.1 Implementation of the Base-Case Relational Schema

The schoolhouse example was created by the DDL seen in Appendix A, which results in the schema described in Appendix C and Figure 16.

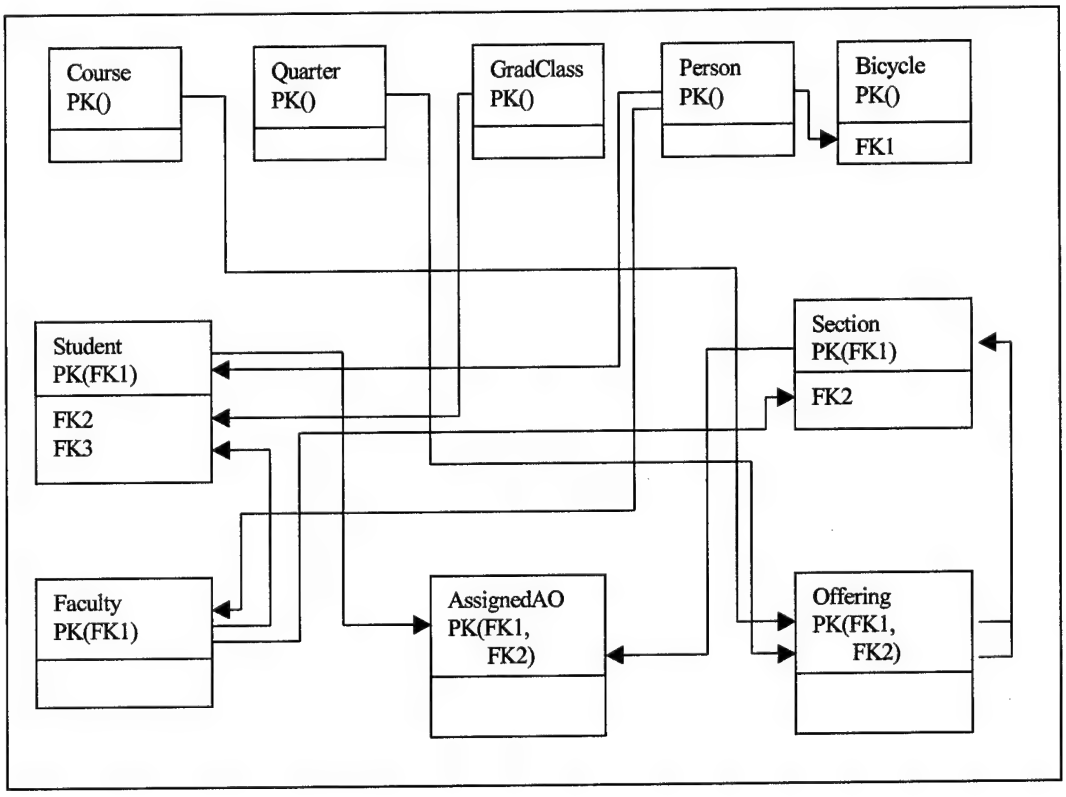

Figure 16: Base-Case Relational Schema--Schoolhouse Example

4.2 Design & Implementation of the Initial Translation Algorithm

4.2.1 Programming Language and DBMS Selection

Java was selected as the application programming language for many reasons. Related research efforts at AFIT use Java™ and the IBM® Visual Age™ integrated development environment extensively [19]. More importantly, the Java Database Connectivity (JDBC™) component gives relatively simple access to Open Database Connection (ODBC) capable databases and their schemas. Access to database metadata and the ability to embed SQL92 queries are essential to this research effort, so Java was the obvious choice. ORACLE® was selected as the DBMS for its ODBC compliance and high availability at AFIT.

4.2.2 Accessing Metadata

The JDBC getConnection() method returns a connection to a specified database. Once the connection is established, the JDBC DataBaseMetadata methods are used to extract the schema as seen in the following discussion.

A database consists of a set of catalogs, each of which contains a set of schemas. In order to access the desired schema for translation, the appropriate catalog and schema must be selected. This is accomplished via the getCatalogs() and getSchemas() methods, which return a set of catalogs for a specified database and a set of schemas for a specified catalog. Once the appropriate schema is identified, getTables() returns table names for the set of relations in the schema. With this set of table names, getPrimaryKeys(), getImportedKeys(), getColumns(), and get IndexInfo() gather the remaining metadata for

each relation. This includes key composition and instance cardinality, as well as the names and descriptions of each attribute, e.g. data type, size, and nullability.

4.2.3 Instance-level Information

Step two of the translation algorithm requires instances of a relation's primary key to be compared with instances of another relation's primary key to identify subset relationships that indicate inheritance relationships. This research accomplished the determination by comparing the cardinality of the intersection of primary keys obtained as seen in Figure 17 from the cardinality of *relation A*. If the cardinalities are equal, then both a subset and generalization relationship exists. Empty relations are filtered from the application of the subset determination algorithm because the intersection of a pair of empty relations always equals the cardinality of the relations, i.e., zero, which causes an erroneous generalization relationship to be discovered.

Let *compatible* define non-empty relations with primary keys that have the same number of columns with identical data types.
Let *cardinality* describe the number of unique instances.
Let *subset* indicate all the values in *A* are also present in *B*.

if compatible(*relation A*, *relation B*)
intersection = (SELECT *primary key of relation A* FROM *relation A*
INTERSECT
SELECT *primary key of relation B* FROM *relation B*)

if *intersection* equals *cardinality of relation A,* then *A* is a *subset* of *B*, unless both relations have the same cardinality.

Figure 17: Subset Determination

4.2.4 Implement Relational and Preliminary OO Representations

The relational and preliminary OO representations discussed in the following two sections provide a mapping from a given relational schema to its OO translation. This mapping can be traversed from a relational to an OO representation or from an OO to a relational representation to retrieve design decisions that aren't captured in the AWSOME translation of the relational schema. Since all schema-level and instance-level metadata is retained in the relational representation and POOR, all design decisions are retained for use by the integrator or any post-processing that may be desired.

4.2.4.1 Representation of the Relational Schema

A wealth of metadata is returned from the JDBC DatabaseMetaData object, and an internal representation is needed to store the results of getDatabaseMetaData() method invocations to preclude unnecessary re-extraction of metadata for subsequent translation steps. A vector populated with instantiations of the object classes depicted in Figure 18 effectively captures the relational schema. The data types shown are the Java types returned by the JDBC method invocations.

4.2.4.2 Preliminary OO Representation

A vector populated with instantiations of the object classes depicted in Figure 19 stores the translated schema.

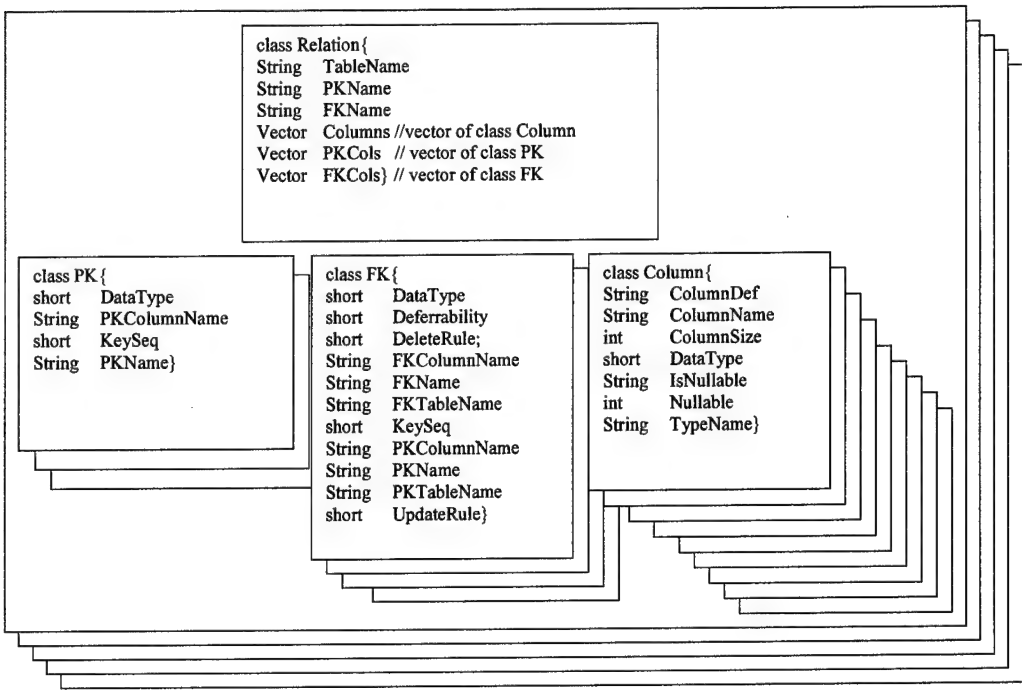

Figure 18: Relational Schema—Internal Representation

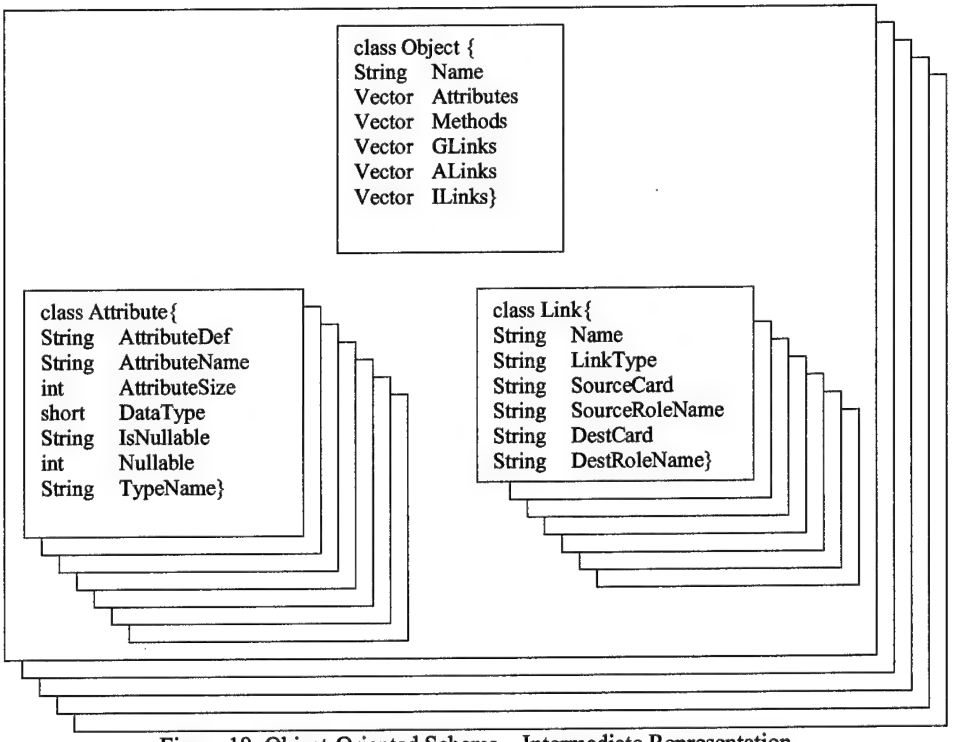

Figure 19: Object-Oriented Schema—Intermediate Representation

4.2.5 Translate the Schema

With the relational schema conveniently captured and a structure for the translated schema in place, the translation algorithm can be applied. As discussed in Section 3.3.2, Sull and Kashyap's translation algorithm produces the sets of object classes and links seen in Figure 9.

4.3 Design and Implementation of the Revised Translation Algorithm

Several issues were discovered when the initial translation algorithm was against the Schoolhouse schema that required resolution. These issues were introduced and analyzed in Section 3.2.5, and the implementation of remedies or extensions to the initial algorithm that result in an improved algorithm are detailed in this section.

4.3.1 Generalization Relationship Detection

Step two of the schema translation algorithm detects generalization relationships among object classes by identifying subset relationships between the primary key instance values of the object classes' source relations. When the relation Person is removed from the schema, Faculty and Student are no longer associated as children of the now abstract `Person` class. Human intervention to identify abstract superclasses during translation step two would simultaneously solve this problem and preclude the research goal of automation. Therefore, pre-existence of all superclasses is essential to complete the generalization discovery in this research.

4.3.2 Avoiding Generation of Redundant or Extraneous Links

There are two methods to avoid generating redundant and extraneous links or associations identified in this research: omission of schema translation step five, and association filtering during AWSOME syntax generation. Schema translation step five requires a domain equivalence determination between components of the primary key of relations with a composite primary key, and the primary key of other relations to generate interaction links. None of the links generated due to domain equivalence determined at the syntactic level provided new schematic information—every generated link was redundant or extraneous. Human intervention would resolve this problem by adhering to a richer concept of domain equivalence, but would violate the research goal of automating translation while preserving translation accuracy. Therefore, omission of step five avoids generating some of the redundant or extraneous links.

The second method of redundant and extraneous link avoidance is association filtering during AWSOME syntax generation. Prior to generating each association, the generateAWSOME() method can detect if a duplicate association between the pair of object classes has already been generated, which can then be filtered from AWSOME syntax generation if desired.

4.3.3 Associative Objects

Relations that are composed exclusively of migrated keys from multiple relations are intersection entities, and relations that also include other attributes are associative entities. Association entities must be modeled as associative objects in an object model, but there

are two equivalent representations for intersection entities: they can be modeled as an association between the parent entities or as an associative object. If intersection entities are modeled as associations, links from the intersection entity must be traced back to the parent objects, and the parent objects' links must be recreated with new destinations and role names. In the case of intersection between two parent entities as seen in Figure 20a, the solution is trivial, but when more than two parent entities are involved as in Figure 20b, the link reconstruction becomes increasingly complex. For simplicity of implementation, this research models both associative entities and intersection entities as associative objects.

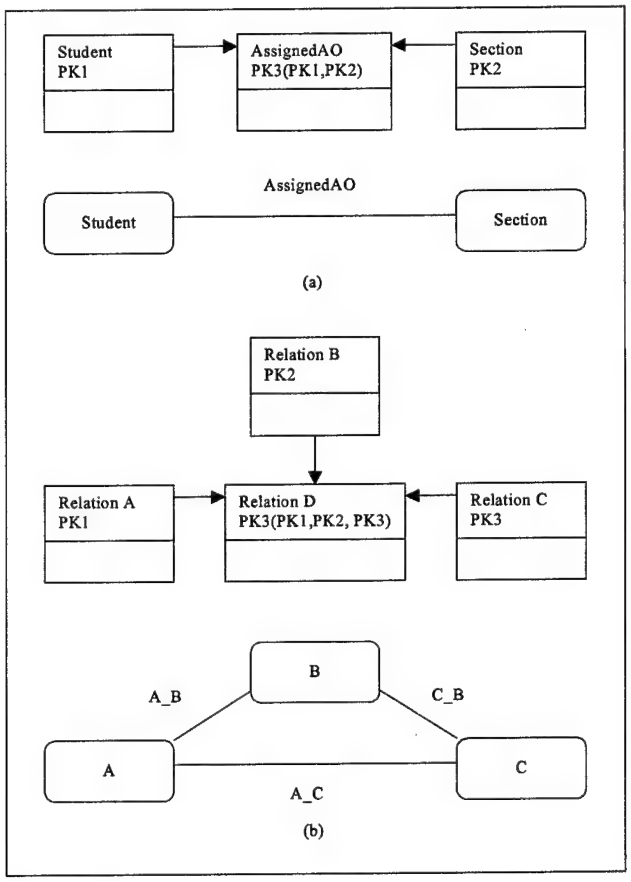

Figure 20: Intersection Entities

4.3.4 Association and Role Naming

An association between Faculty and Student can be named Teaches or Learns with equal validity. The Teaches association has Faculty as the association anchor and Learns has Student as the anchor. This arbitrary selection of the anchor entity lends an implied direction to an association, and this research found the implied direction of generated associations differed from the original schoolhouse example approximately half the time. Since key migration is the essential mechanism used to generate associations and keys are traced from child to parent entities, the child entity is always selected as the association anchor in this research. Consistently selecting the child entity as the association anchor is less arbitrary than a complete absence of consistency in anchor selection.

Association names are implemented as a concatenation of the associated objects' names with the anchor object first, e.g. Student_Faculty. Subsequent associations identified between the same objects will append a number for differentiability, e.g., Student_Faculty_2. Role names are a concatenation of the objects' names and migrated attributes names, e.g., Student _AdvisorToFaculty_SSN. More semantically enriched association and role names could be devised with human intervention, but this implementation is the intended automated solution.

4.3.5 Modeling Abstract Object Classes

No construct currently exists to model abstract object classes in the AWSOME metamodel, so all classes are implicitly concrete [17]. Ideally, a Person should never be instantiated except as a Faculty or Student object that extends the Person class. This

research doesn't discover abstract classes due to reliance on key migration for domain equivalence determination. However, if an abstract class were discovered, there would be no way to explicitly declare it as such in the current state of AWSOME syntax.

4.3.6 Results

Application of the revised translation algorithm with and without association filtering results in the OO schemas depicted in Figures 21 and 22 respectively, and the equivalent AWSOME syntax seen in Appendices E and F. Object model components are generated into AWSOME syntax in the following order: types, sets, classes, derived classes that incorporate inheritance relationships, aggregation relationships, and finally interactions.

A generalization relationship between Offering and Section was discovered during translation, though the relationship is actually an aggregation in the relational schema that was translated from a *one to m* association in the original OO schema. Similar duplication exists between `Person` and its children, though its generalization relationships are valid. In the case of redundant associations, if one of the associations is a generalization, then retaining the redundant association prevents possible mischaracterization of the association by filtering a correct association from being generated. If the first association is generated and subsequent associations are filtered, the order of AWSOME syntax generation lends itself to a preference toward preserving generalizations over aggregation or interactions, which would mischaracterize the Offering to Section association as a generalization. If the preference were reversed, then the Person to Faculty and Person to Student associations would be mischaracterized as interaction or aggregation. Since this research couldn't

devise an automated algorithm that can correctly determine which association to filter in every case, it retains redundant associations if correct association filtering can't be assured.

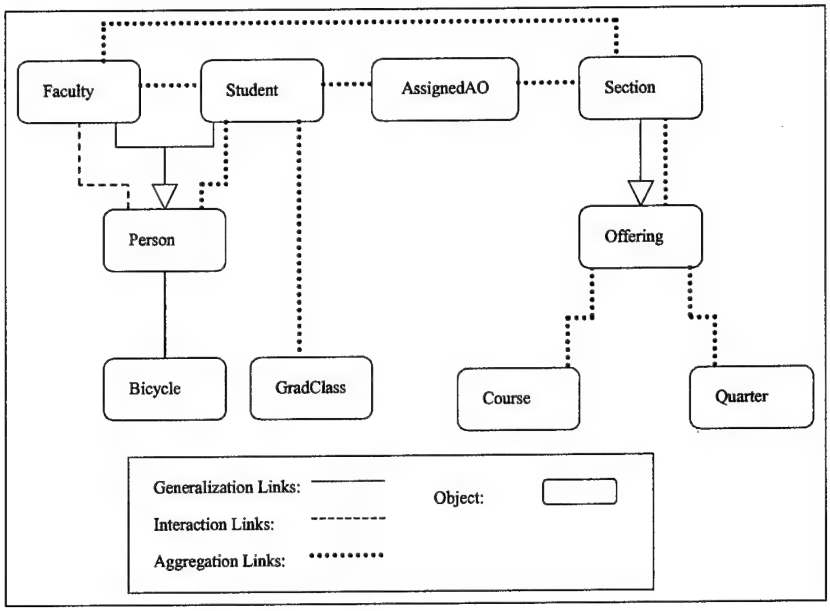

Figure 21: Revised Algorithm Results Without Filtering

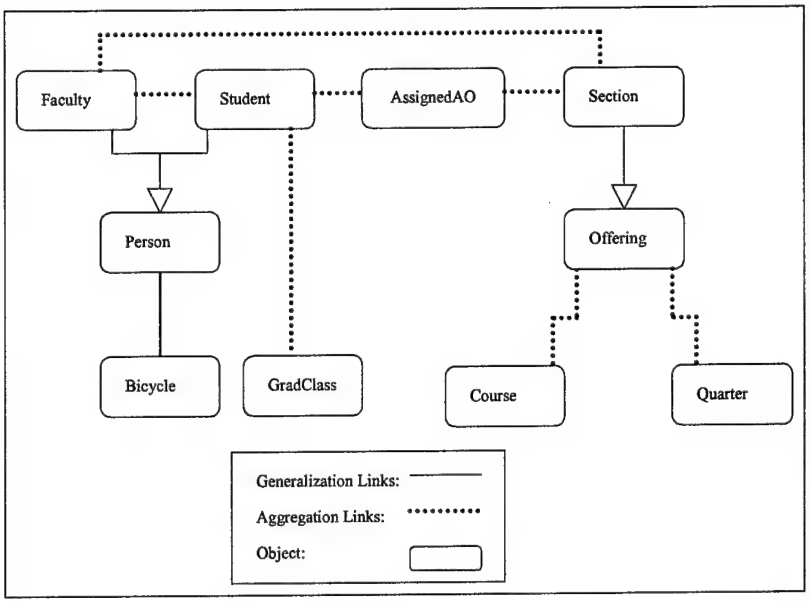

Figure 22: Revised Algorithm Results With Association Filtering

4.4 *Validation Case: CLASPICS*

4.4.1 Validation-Case Schema

Computerized, Lightweight Assistant for Student Program Identification and Course Selection, CLASPICS, is a web-enabled course scheduling system designed and implemented at AFIT [18]. It was selected as the validation case for this research primarily because it is readily accessible and is an operational system rather than a synthetic example. The relational schema is depicted in Figure 23 and is fully described in Appendix B. The relations depicted are permanent members of the schema, while others are dynamically created while the CLASPICS system is being utilized. Some of the dynamically created relations are transient lookup tables or simple lists, e.g. RequestedCourseList, while others remain until the CLASPICS database is recreated, e.g., EN_TABLE.

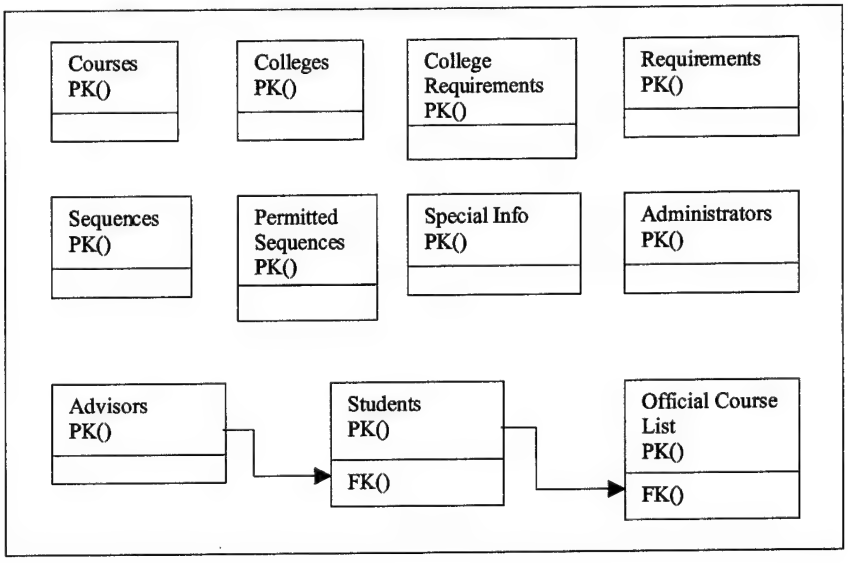

Figure 23: CLASPICS Relational Schema

4.4.2 Results

The OO translation of the Computerized Lightweight Assistant for Student Program Identification and Course Scheduling (CLASPICS) relational schema that resulted from application of the revised translation algorithm is depicted in Figure 24. All permanent relations were discovered and translated into object classes except for Courses. The dynamically created relation EN_TABLE was also correctly discovered and translated. All three key migrations present in the schema were correctly discovered and translated into associations.

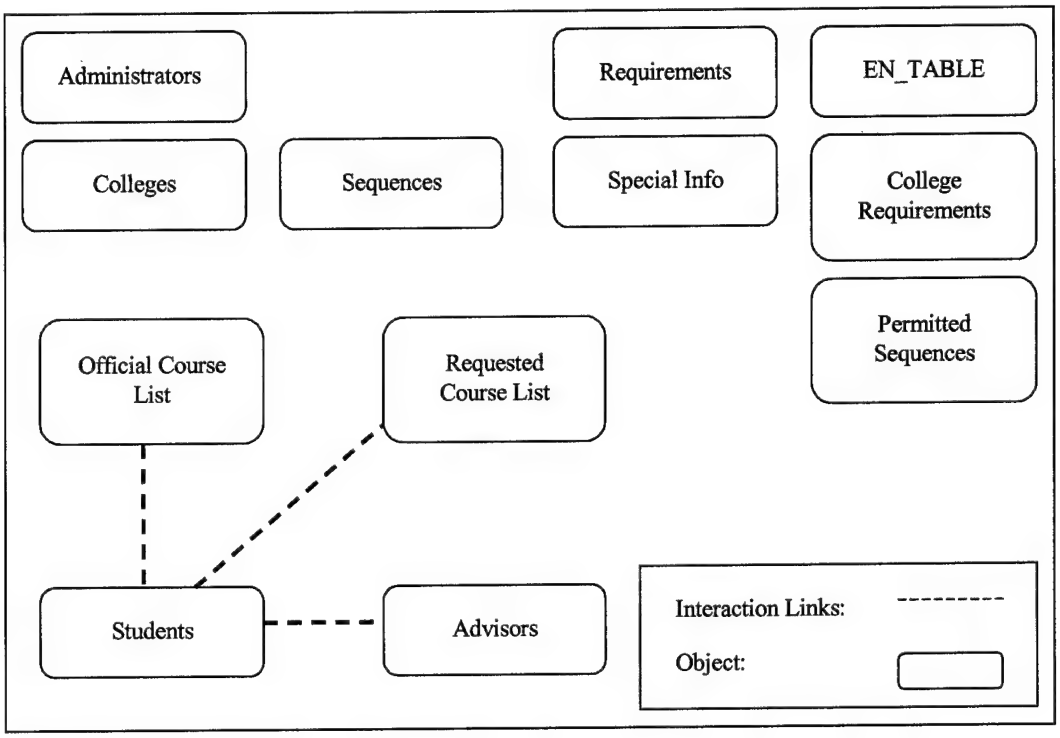

Figure 24: CLASPICS OO Schema Resulting from Application of Revised Translation Algorithm

4.4.2.1 Chubby-Keyed Wallflower Relations and Dynamically-Created Tables

In order for an object class to be created by the translation algorithm, it must have a single-attribute primary key or participate in a key migration. The CLASPICS relational schema includes the Courses relation, which has a composite primary key but isn't involved in a key migration, and therefore isn't translated into an object class. This research dubbed such relations as "chubby-keyed wallflower relations," and resolved this problem via the createWallflowerObjects() method, which creates object classes for relations that meet the wallflower criteria just prior to AWSOME syntax generation. The final algorithm that includes createWallflowerObjects()is seen in Figure 25, and the result when applied to the CLASPICS schema is seen in Figure 26.

CLASPICS dynamically generates tables during operation, some of which are non-transient, explicitly keyed relations and some of which are simple lists or "scratch pads" that store temporary results of user operations that are later deleted. Explicitly keyed relations that expand the schema should be detected and incorporated into an updated translation. Conversely, non-explicitly keyed tables are transient members of the schema and may possibly contain duplicates; therefore, they should be ignored. Validation of the translation algorithm against CLASPICS showed that explicitly keyed relations are captured and translated into object classes by the translation algorithm, e.g., EN_TABLE, while non-explicitly keyed tables are ignored.

1. *Select a complete subset of relations from a relational schema for translation.*
2. Create corresponding object classes for each relation with a single-attribute primary key.
3. Create generalization links.
   For all relations R1 and R2
   If the instance values of R1's PK are a subset of the instance values of R2's PK, then R2 is a subclass of R1
   If associated object classes for R1 and R2 don't exist, create them
   Create a generalization link between O1 and O2
   Determine and assign link name and cardinality:
   source cardinality is zero-to-one and destination cardinality is zero-to-n.
4. Create interaction links between single-foreign-keyed relations and their parent.
   For all relations
   If R1 is single-FK'd, then R2 is the parent relation
   If associated object classes for R1 and R2 don't exist, create them
   Create interaction links between R1 and R2
   Determine and assign link name and cardinality
   If child is non-unique and the child's FK cardinality is less than parent's PK cardinality, then source cardinality is zero-to-n and destination cardinality is zero-to-one.
   If child is non-unique and the child's FK cardinality equals the parent's PK cardinality, then source cardinality is one-to-n and destination cardinality is one-to-one.
   If child is unique and the child's FK cardinality is less than the parent's PK cardinality, then both the source and destination cardinalities are zero-to-one.
   If child is unique and the child's FK cardinality equals the parent's PK cardinality, then both the source and destination cardinalities are one-to-one.
5. Create aggregation links between multiple-foreign-keyed relations and their parents
   For all relations
   If R1 is multiple-FK'd, then R2..Rn are the parent relations
   If associated object classes for R1..Rn don't exist, create them
   Create aggregation links between R1 and R2…R1 and Rn
   Determine and assign link name and cardinality as in step 4.
6. Create interaction links between composite-primary-keyed relations and object classes with syntactically equivalent domains.
   For all relations
   If R1 is composite-PK'd, then R2...Rn have symantically-equivalent PK's to column(s) in the PK of R1
   If associated object classes for R1..Rn don't exist, create them
   Create interaction links between R1 and R2…R1 and Rn
   Determine and assign link name and cardinality
7. *Create corresponding object classes for each relation with a composite-attribute primary key and no imported or exported keys.*
8. *Generate AWSOME output--types, object classes, subclasses and associations.*

Figure 25: Final Translation Algorithm: Including Chubby-Keyed Wallflower Detection and Object Class Creation

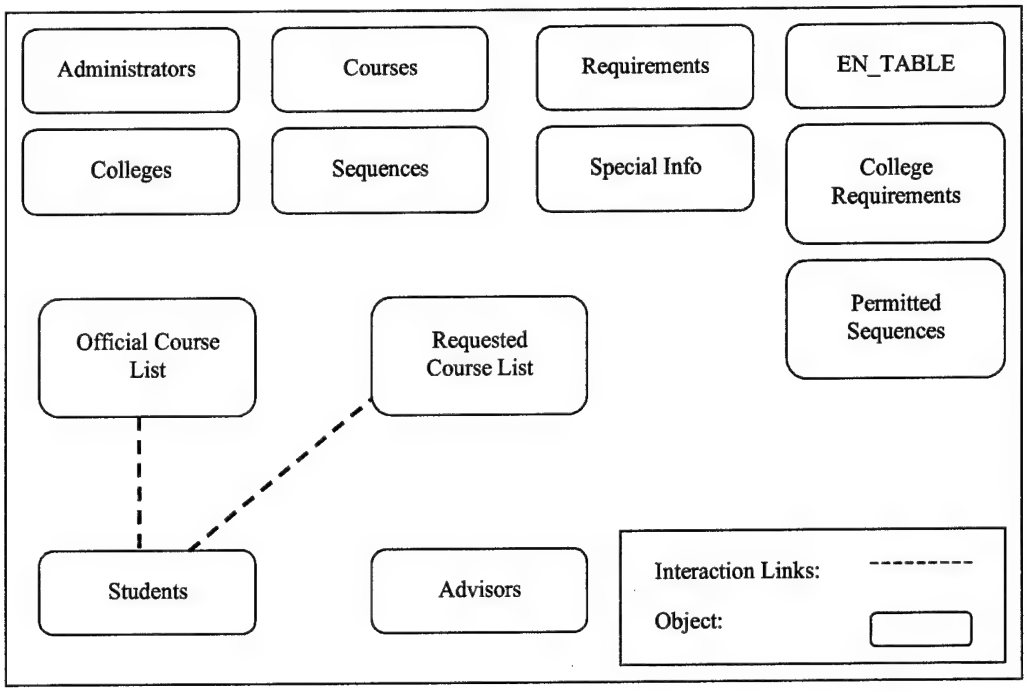

Figure 26: CLASPICS OO Schema Resulting from Application of Final Translation Algorithm

4.4.3 Revalidation on Schoolhouse

The final translation algorithm was also revalidated against the Schoolhouse example with the result seen in Figure 22. No change was expected, as no chubby-keyed wallflowers are present in the Schoolhouse schema, and no side effects were identified.

4.5 AWSOME Syntax Validation

A concurrent research effort [17] produced a tool that parses AWSOME syntax and produces abstract syntax trees that model the translated schema in the AWSOME metamodel. The AWSOME syntax that was generated by the improved and final

translation algorithms applied against the base and validation schemas, e.g., Appendices C and D, was successfully parsed into the AWSOME metamodel by [17] without error.

4.6 Schema Change Detection and Revised Translation Generation

Two main approaches to schema change detection and revised translation generation were discussed in Chapter 3, event-driven and temporal. Both approaches are analyzed in this section, and a hybrid approach that applies both techniques is introduced and discussed. After analysis, the tunable-temporal approach was selected for implementation and validation, while the event-driven and hybrid approaches are merely analyzed.

4.6.1 Event-driven Approach

The event-driven approach can be implemented by defining triggers on appropriate commands, e.g., UPDATE, CREATE, DROP, or ALTER TABLE, and DROP or UPDATE COLUMN. The trigger body would be written in PL/SQL, which is ORACLE's extension to SQL that includes higher-order program constructs. Either the DBMS_PIPE or HOST command can be used to execute an external program, in this case the schema change detection and re-translation program. The problem with the event-driven approach, above those discussed in Chapter III, is its lack of portability—this capability doesn't work on all operating systems [13]. Since a proprietary, operating system-specific extension to PL/SQL is required to invoke an external program, new DBMS application code must created for every database type, and worse, some databases may not support triggered execution of external programs. Portability is further hampered if an inability to call

external programs is solved via migration of the entire translation program to the DBMS, which would further increase the differences between each implementation. Schedule limitations and lack of portability induced this research to forego an event-driven approach to schema change detection and retranslation in favor of a tunable-temporal approach.

4.6.2 Tunable-temporal Approach

The tunable-temporal approach can cause sub-optimal translation currency and unnecessary re-applications of the translation algorithm. Since the translation algorithm is only re-applied at specified intervals, highly dynamic schemas result in poor translation currency when the selected interval is long. In relatively static environments, the algorithm is unnecessarily re-applied even if no schema change has occurred.

However, the benefits of the tunable-temporal approach include saturation avoidance and high portability. The integrator can decide how often to re-integrate and can set an algorithm re-application interval that doesn't permit saturation. After an interval specified in the loiter() method, the relational schema is re-extracted and compared against the previously translated relational schema by the detectSchemaChange() method. The original and new Relations vectors are compared, and an updated translation is generated only if a schema change is detected. This re-extraction, comparison, and deterministic re-translation cycles as long as the translation application is running. High portability is attained because there is no need for implementation-specific triggered procedures and the algorithm re-application logic is included in the translation application.

4.6.3 Hybrid Approach

If an event-driven approach for schema-level change detection was combined with a temporal approach for instance-level change detection, a hybrid approach could be devised. A hybrid approach that capitalizes on the strengths of each component approach and minimizes the weaknesses of each component approach would be desirable. Since schema-level change is very infrequent compared to instance-level change, schema level change could be detected by an event-driven solution, and instance-level change could be detected by a temporal solution. This could mitigate the risk of saturation seen in the event-driven approach that is caused by frequent insertions and deletions of tuples. At the same time, the hybrid approach ensures schema-level changes, e.g., added or dropped relations immediately trigger a schema re-translation, which improves translation currency over the temporal approach. The portability concerns associated with the event-driven approach are still evident in the hybrid approach.

4.6.4 Validation

As discussed in Section 4.6.2, the tunable-temporal approach to schema change detection and translation regeneration was implemented in this research. The validation protocol and results are presented in detailed as Attachment E. Validation against the Schoolhouse schema included detection and consequent translation regeneration for addition and removal of relations, addition and modification of columns, and instance-level tuple deletion causing association cardinality to change.

V. Conclusions and Recommendations

This research provides the largely automated, extensible solution for translating heterogeneous schemas to an OO representation seen in Figure 27. A majority of the translation tasks from relational to OO schemas was successfully automated, with filtering a portion of the redundant associations postponed for pre-integration, as discussed in Section 4.3.6.

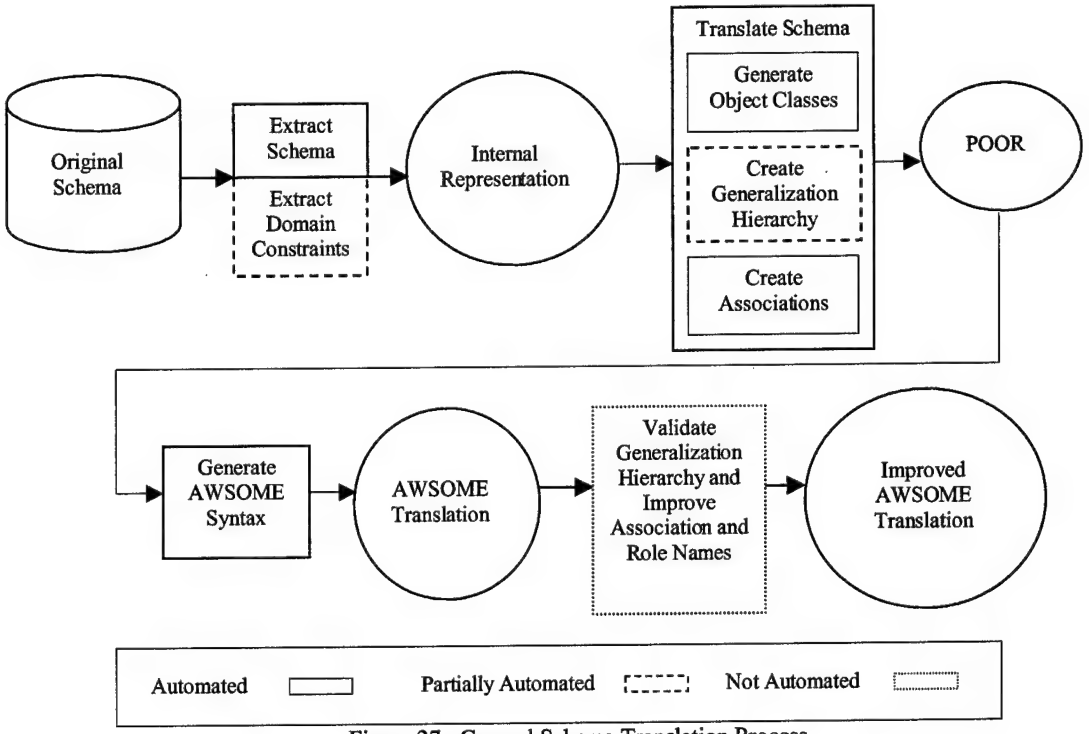

Figure 27: General Schema Translation Process

This research did not produce an automated solution for XML file translation as a validation of extensibility of this methodology, but this is strictly due to schedule limitations rather than problem difficulty.

5.1 Results

This research effort produced a highly automated, significantly improved relational to OO schema translation algorithm that was derived from the initial schema translation algorithm proposed by Sull and Kashyap [5]. The initial algorithm was tested against the Schoolhouse relational schema generated by a related AFIT research effort [12]. Several modifications to the initial algorithm resulted in an improved translation algorithm that was then validated against the CLASPICS relational schema [18], which is an operational course scheduling system used at AFIT. Validation on CLASPICS identified another deficiency in the initial algorithm that was corrected and re-validated on Schoolhouse and CLASPICS.

Several improvements to the initial schema translation algorithm were accomplished in this research, which produces a translation that is a far better candidate for integration. The initial translation algorithm produced a hybrid translation consisting of relations, objects, and three types of inter-entity links, which is unsuitable for schema integration efforts that require a common OO data model, e.g., Ashby [20]. The improved algorithm produces an OO translation in AWSOME syntax [17] that consists of object classes, derived object classes, and inter-entity associations. The initial algorithm assumed domain constraint availability and didn't explicitly address semantic equivalence between attributes, which resulted in numerous extraneous inter-entity links that required post-translation resolution. Even with incomplete availability of domain constraints, the improved algorithm guarantees semantic equivalence by tracing key migrations between

relations, which greatly reduces the number of extraneous links requiring later resolution. In the improved algorithm, the only redundant links left for post-translation resolution are those that lack guaranteed-correct, automated resolution. Finally, the improved algorithm also corrects a failure to translate relations that have composite primary keys and don't participate in key migration.

The improved schema translation algorithm also achieved a high level of automation and portability. Event-driven and temporal approaches to schema change detection and revised translation generation were evaluated, and a tunable-temporal was implemented. After the user selects the database and schema to translate, the algorithm automatically produces a revised OO translation for integration at user-specified intervals, if a schema change occurs. High portability was achieved by using Java JDBC and avoiding non-standard extension to SQL. The translation algorithm can be used against any relational schema on an ODBC capable database.

5.2 Conclusions

This section provides a characterization of the feasibility of automating various aspects of schema translation based on lessons learned during design and implementation of an automated schema translation application in this research.

5.2.1 Availability of Domain Constraints

An essential assumption of Sull and Kashyap's methodology introduced in Section 2.6 is the availability of domain constraints for domain equivalence determinations during

schema translation. In relational implementations, the DBMS provides a finite set of data types for use by applications. The schema is described by these types, but domain constraints within these types that are enforced by applications are not available via standardized schema extraction methods because they reside in the application. Since each application is unique, no automated solution could be devised to extract domain constraints across applications. Estimation of domain constraints via instance-level examination can be automated, but it only describes the range of values currently in a database rather than those intended by the designer. Reliance on domain constraints inferred from instance-level examination enables erroneous conclusions to be drawn, e.g., a generalization can be discovered and later voided by an added instance.

Migrated keys, other captures of the designer's intent, or human domain expertise is required to ensure semantic equivalence. In general, the more of a designer's intent that can be captured in the schema versus the application, the less operator intervention is required for accurate schema translation. This research mitigated incomplete availability of domain constraints by relying on key migration to assure semantic equivalence, but this technique or an analog may not be available in other data models. Therefore, domain constraint extraction should be performed before schema translation to prevent erroneous assumptions from being reflected in the translated schema. Access to the application code, accurate data description documentation or domain expertise is required if domain constraints are to be refined beyond the general types offered by the data source.

5.2.2 Association and Role Name Assignment

A fully automated algorithm for assigning association and role names that are as semantically rich as those assigned by a human expert could not be devised by this research; however, a semi-automated solution was successfully implemented. The automatically generated names concatenate the names of entities and attributes participating in each association. For example, the association between Student and Faculty is named Student_Faculty, with roles STUD_RADVISES__FAC_PERSON_SSNtoFAC_PERSON_SSN and FAC_PERSON_SSNtoSTUD_RADVISES__FAC_PERSON_SSN. This information helps a human expert to infer correct, semantically rich association and role names but is semantically insufficient to serve as the final translation. Even with appropriate, semantically rich association and role names assigned, the anchor entity for an association may differ between a pair of schemas to be integrated. For example, an association between Faculty and Student may be named Teaches in one schema and Learns in another. This disparity would likely require human intervention for resolution, so prior to or during integration, a human expert must choose the desired anchor entity and transform the automatically generated names into appropriate association and role names.

5.2.3 Generalization Relationship Detection

Automated identification of inter-entity generalization relationships was partially successful--this research relies on the pre-existence of parent entities and instance-level examinations to identify generalizations. Specifically, if the instances of the primary key of one relation are a subset of the instances of the primary key of another relation, then a

child to parent generalization relationship is identified automatically. An automated algorithm to identify and create abstract superclasses was not discovered in this research. If the parent entity does not already exist, then the subset determination cannot be made, and the consequent generalization is not discovered. Reliance on instance-level examination allows erroneous generalizations to be discovered that can be later invalidated by adding an instance of the child entity that isn't a subset of the parent. Though the automated translation application mitigates this problem by reapplying the translation algorithm and updating the generalization hierarchy, failure to identify and create abstract superclasses makes the automated solution a poor substitute for human expertise. After the automated translation application has run, a human expert can create necessary abstract superclasses and restructure the generalization hierarchy to generate an improved OO translation, as depicted in Figure 27.

5.2.4 Design Decisions Lost or Captured

As discussed in Section 4.3.6, confusion arises on how to model associations that can't be resolved via automation. The *one to many* association TaughtAs in Figure 4 can be modeled in a relational schema as a migrated key from Offering to Section, or as another relation. The design decision made during translation causes semantic loss that can't be later retrieved. Loss of design decisions is especially prevalent when translating from a stronger to a weaker data model. If all design decisions are captured, then the object model is implied to exist and the translation effort would be trivial. Only instances of more than one Section for a given Offering discovered during translation can determine that more than one Section can exist for a given Offering. Again, reliance on instance-level information

may allow invalid assumptions to be made. Design decisions made by the automated translation application produced by this research are captured in the POOR described in Section 4.2.4.2, and can be referenced by a human expert attempting to refine the OO translation.

5.2.5 Translation Algorithm Improvement

As discussed in Chapters 3 and 4, several improvements were made to the base translation algorithm. Inability to translate composite-keyed relations that don't participate in key migrations (chubby-keyed wallflowers) was identified and resolved in Section 4.4.2.1. Migrated keys are used to automate semantic equivalence determination whenever available, rather than the less reliable and hard to automate concept of "domain equivalence" used in the initial algorithm. This research produces an OO representation with association redundancy automatically removed whenever translation correctness can be assured. Minimal pre-integration restructuring is required, which is a distinct improvement over the hybrid schema with highly redundant associations produced by the initial algorithm.

5.2.6 Schema Change Detection and Revised Translation Generation

Event-driven and *tunable-temporal* approaches to schema change detection and revised translation generation were evaluated in this research. The event-driven approach re-applies the translation algorithm when an event likely to cause a schema change is detected, while the tunable-temporal approach re-applies the translation algorithm at pre-specified intervals. The benefit of the event-driven approach is minimized translation

maintenance cost to achieve maximum translation currency, however, it lacks portability and has a propensity for system saturation. In the worst case, the translation application would generate another translated schema as quickly as the application can cycle. The tunable-temporal approach can cause sub-optimal translation currency and potentially wasteful re-applications of the translation algorithm, but it doesn't have the saturation and portability problems associated with the event-driven approach. The tunable-temporal approach should be the default choice for schema change detection and re-translation, while the event-driven approach is indicated only in the event that maximized translation currency is worth the cost of designing implementation-specific stored procedures.

5.2.7 Solution Portability

The relational to OO schema translation application produced by this research is platform independent and can be easily applied to any relational schema residing in an ODBC compatible database. The three simple steps outlined in Figure 28 will apply the translation application against a given schema.

5.2.8 Solution Extensibility

We propose that the automated schema translation application and methodology generated by this research can be extended by tailoring the generic steps in Figure 27 to the data model of the data source schema to be translated. A solution for XML files was not completed as initially planned, which would have validated the extensibility of this approach. However, we propose a solution for this, or any other, data source type could be

developed by following the steps in Figure 27. An existing internal representation, e.g. W3C's Document Object Model or another analog for the JDBC databaseMetaData

1. *Customize ExtractMetaData() Method*

   Enter the appropriate url, e.g.

   ```
   String url = "jdbc:oracle:thin:@platformName.afit.af.mil:1521:databaseName"
   ```

   Enter a valid userName and password for the selected database, e.g.

   ```
   String userName = "myName";

   String password = "myPassword";
   ```

   Enter a appropriate schema and catalog for the desired schema which can be obtained via the ExtractMetaData.getSchemas() and getCatalogs() methods, e.g.

   ```
   String schema = "mySchema";

   String catalog = "myCatalog";
   ```

2. *Customize Loiter() Method*

   Enter the appropriate schema change detection and update propagation interval in seconds, e.g.

   ```
   int numSeconds = 7200
   ```

   iterates the change detection and update propagation algorithm every two hours.

3. *Execute ExtractMetaData()*

Figure 28: Application Porting Procedure

object, can be directly substituted for the relational internal representation currently in place. If an exiting representation doesn't exist for a given schema type or is determined to be inadequate, then it must be created or derived from an existing representation. Once an internal representation is in place, the translation algorithm must be tailored to convert the schema type into POOR. In this way, a new module for translating a schema type only has to be devised once to provide a general solution for schemas of that type.

Several factors will affect the level of automation achievable for each schema type. In the relational to OO solution, incomplete access to domain constraints was circumvented by relying on key migration to assure semantic equivalence. The level of structure inherent in the source data model and its capacity to capture semantics are crucial—the relational model is highly structured and semantically powerful, while HTML files are relatively unstructured and XML files lie somewhere in between. Standardization is also key to devising general solutions—adherence to SQL92 and ODBC standards facilitates development of general automated solutions for relational to OO translation. The emerging W3C standard for XML improves the opportunity for a general automated solution for XML to OO translation as well.

5.3 Recommendations for Further Research

5.3.1 Improve the Current Translation Application

Triggers, stored procedures, and update and delete rules provide additional schematic information that can improve translation accuracy. Automated capture of triggers and stored procedures was not addressed in this research, as they weren't present in the base or validation schemas, but further research needs to be accomplished to determine whether their translation can be automated.

Intersection entities are modeled as associative objects in the OO translation produced by the translation application. An algorithm can easily be devised to detect

intersection entities and model them as pure associations rather than as an object with links to the associated entities.

AWSOME should implement the ability to model abstract classes, and research into automated detection and creation of abstract superclasses should be pursued. Abstract superclass detection and creation currently requires human intervention, but techniques in automated semantic equivalence determination may lessen the user intervention required.

5.3.2 Validate Extensibility and Utility of the Methodology

Applying the methodology in Figure 27 to another data source type would validate the extensibility of this research. XML is an excellent candidate because analogs for the JDBC databaseMetaData object are readily available as freeware on the W3C home page. Validation of the utility of this methodology to integration efforts is also desirable. This can be achieved by applying Ashby's integration methodology to the output of the automated schema translation application.

5.3.3 Apply Artificial Intelligence Techniques to the Problem Domain

Artificial intelligence techniques promise the ability to learn, reason over uncertainty, share ontologies, and improve semantic reasoning. Application of these techniques to the schema translation problem, particularly the unresolved translation tasks, is a fertile area for research. Rather than treating every translation as a new problem, an agent may be able to capture knowledge from previous translation efforts into a knowledge base that can be applied to the translation effort. An agent based "assistant" could be devised to help a human translator/integrator through the translation/integration process.

Appendix A. Schoolhouse Relational Schema DDL

This is the code that creates the Schoolhouse relational schema used as the base case in this research.

```
CREATE TABLE GradClass (
program          VARCHAR2(20),
year          INTEGER,
graddate          DATE,
desig          VARCHAR2(6) NOT NULL,
PRIMARY KEY (desig) )

CREATE TABLE Course (
ctype          VARCHAR2(4),
cnum          INTEGER,
ctitle          VARCHAR2(40) NOT NULL,
cdesc          VARCHAR2(80),
creditHours          INTEGER,
lectureHours          INTEGER,
abetDes          INTEGER,
abetSci          INTEGER,
abetMath          INTEGER,
abetOther          INTEGER,
PRIMARY KEY (ctitle) )

CREATE TABLE Person (
lastname          VARCHAR2(20),
midinitial          VARCHAR2(1),
firstname          VARCHAR2(20),
birthdate          DATE,
ssn          VARCHAR2(9) NOT NULL,
height          INTEGER,
weight          INTEGER,
PRIMARY KEY (ssn) )

CREATE TABLE Bicycle (
bikeSN          VARCHAR2(9) NOT NULL,
Owns__Person_ssn          VARCHAR2(9),
FOREIGN KEY (Owns__Person_ssn) REFERENCES Person( ssn),
PRIMARY KEY (bikeSN) )
CREATE TABLE Fac (
academicRank          VARCHAR2(3),
Person_ssn          VARCHAR2(9) NOT NULL,
FOREIGN KEY (Person_ssn) REFERENCES Person( ssn),
PRIMARY KEY (Person_ssn) )
```

```
CREATE TABLE Quar (
qname          VARCHAR2(10) NOT NULL,
year          INTEGER,
qstart          DATE,
qend          DATE,
PRIMARY KEY (qname) )

CREATE TABLE Stud (
gpa          INTEGER,
Person_ssn          VARCHAR2(9) NOT NULL,
MemberOf__GradClass_desig          VARCHAR2(6),
RAdvises__Fac_Person_ssn          VARCHAR2(9),
FOREIGN KEY (Person_ssn) REFERENCES Person( ssn),
FOREIGN KEY (MemberOf__GradClass_desig) REFERENCES GradClass( desig),
FOREIGN KEY (RAdvises__Fac_Person_ssn) REFERENCES Fac( Person_ssn),
PRIMARY KEY (Person_ssn) )

CREATE TABLE Offrng (
code          VARCHAR2(9),
Course_ctitle          VARCHAR2(40) NOT NULL,
Quar_qname          VARCHAR2(10) NOT NULL,
FOREIGN KEY (Course_ctitle) REFERENCES Course( ctitle),
FOREIGN KEY (Quar_qname) REFERENCES Quar( qname),
PRIMARY KEY (Course_ctitle , Quar_qname) )

CREATE TABLE Sect (
snumber          INTEGER,
TAs__Offrng_Course_ctitle          VARCHAR2(40) NOT NULL,
TAs__Offrng_Quar_qname          VARCHAR2(10) NOT NULL,
Teaching__Fac_Person_ssn          VARCHAR2(9),
FOREIGN KEY (TAs__Offrng_Course_ctitle , TAs__Offrng_Quar_qname)
REFERENCES Offrng( Course_ctitle , Quar_qname),
FOREIGN KEY (Teaching__Fac_Person_ssn) REFERENCES Fac( Person_ssn),
PRIMARY KEY (TAs__Offrng_Course_ctitle , TAs__Offrng_Quar_qname) )

CREATE TABLE AssignedAO (
Stud_Person_ssn          VARCHAR2(9) NOT NULL,
Sect_TAs__Offrng_Course_ctitle          VARCHAR2(40) NOT NULL,
Sect_TAs__Offrng_Quar_qname          VARCHAR2(10) NOT NULL,
FOREIGN KEY (Stud_Person_ssn) REFERENCES Stud( Person_ssn),
FOREIGN KEY (Sect_TAs__Offrng_Course_ctitle ,
Sect_TAs__Offrng_Quar_qname) REFERENCES Sect( TAs__Offrng_Course_ctitle
, TAs__Offrng_Quar_qname),
PRIMARY KEY (Stud_Person_ssn , Sect_TAs__Offrng_Course_ctitle ,
Sect_TAs__Offrng_Quar_qname))
```

Appendix B. CLASPICS Relational Schema DDL

This is the code that creates the CLASPICS relational schema used as the validation case in this research.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.io.*;


//-------------------------------------------------------------------------
----------------------------//
//
//
//            generateDatabase.java
//
//
//
//            Written by Jason Gunsch,     June 1999
//
//              Last update : July 26, 1999   : Commenting
//
//
//
//            This File contains class definitions for the following
classes : generateDatabase      //
//
//
//-------------------------------------------------------------------------
----------------------------//


//-------------------------------------------------------------------------
----------------------------//
//
//
//            generateDatabase class definition
//
//
//
//            Written by Jason Gunsch,     June 1999
//
```

```
//              update : July 15, 1999  : Commenting
//
//              Last update : Aug.3, 1999 : update table descrptions,
added the tables Advisors     //
//                                          and Administrators.
//
//
//
//            This is an administrative tool used to generate a new
database if there is not        //
//              one already, or to reinitialize tables that may have
been lost for some reason.     //
//              It is a safe function, in that it will not overwrite
tables that are already there  //
//              under that name.  It operates as a function that is
called by another program.      //
//              When it is called, the Connection object and Statement
object must be passed to     //
//              it. (Actually, the Connection is not currently
required, but may be in the the      //
//              future)  The tables it generates are as follows :
//
//
//
//        Courses, Colleges, CollegeRequirementsTables,
RequirementsTable, SequencesTable,          //
//        PermittedSequencesTable, SpecialInfo, Students,
OfficialCourseList, RequestedCourseList   //
//        Advisors, Administrators
//
//
//
//-------------------------------------------------------------------
----------------------------//

public class generateDatabase {

  private Connection conn; // The Connection object passed from the main
program.
  private Statement stmt; // The Statement object passed from the main
program.
  private boolean successful;



/* generateDatabase
*  Default Constructor.
*  parameters : Connection conn - the Connection object
*  : Statement stmt - the Statement object
*  This function is the constructor and the entirity of the class.  It
accepts as parameters the Connection and Statement and then tries to
generate
```

```
*    the basic tables in the database, the ones necessary to run the
Database Setup tools.  It uses the basic JDBC and SQL commands to
interface with
*    the database.  It is pretty much self explanitory if you
understand JDBC and SQL.
*/

  generateDatabase (Connection conn, Statement stmt, String user,
String password) {

    this.conn = conn;
    this.stmt = stmt;

    System.out.println("\n");

            // this next section only executes if the program is called
            // by the cleanstart program.  It attempts to create the
            // program's defult user.  You can change the tablespace,
user
            // name, and password.  To change the tablespace all that
            // needs to be done is to change the sql code here and
            // recompile.  To change the id and password you need to
            // change this sql code and also go into the programs
            // DBAdmin.java, register.java, and under the
setDriverInformation
            // section set dbproductname to the new id, and
dbdriverversion to
            // the new password.  The names of the variables are
intentionally
            // misleading.


   if(!user.equals("programadminverified")) {
    try {
      stmt.executeUpdate("create user or8regdfu identified by
oX3Fc83AD95vMX20u93P4b5e default tablespace registration");
      stmt.executeUpdate("grant connect to or8regdfu");
      stmt.executeUpdate("grant resource to or8regdfu");
    } catch (SQLException ex) {
      if(ex.getMessage().equals("ORA-01920: user name 'OR8REGDFU'
conflicts with another user or role name") ) {
        System.out.println("Programs already registered with the
database, or user name is taken.");
      } else if(ex.getMessage().equals("ORA-01031: insufficient
privileges") ) {
        System.out.println("You do not have sufficiant database
privileges to install these programs!  "+
          "Please see your database administrator.\n\n");
        System.exit(0);
      } else {
        System.out.println("Registering program with database :
"+ex.getMessage());
      }
    }
```

```
    }

     // Generating Table Courses

       try {
         stmt.executeUpdate("create table or8regdfu.Courses (prefix
CHAR(4) not NULL, codenum CHAR(3) not NULL, title VARCHAR(100) "+
           "not NULL, dept CHAR(3), cdescr VARCHAR(10) not NULL,
creditmin INTEGER not NULL, creditmax INTEGER not NULL, prereq "+
           "VARCHAR(100) not NULL, coreq VARCHAR(100) not NULL, descr
VARCHAR(2000) not NULL, quarters VARCHAR(15) not NULL, "+
           "repeatable CHAR(1) not NULL, primary key(prefix,
codenum))");
         System.out.println("Courses Created");
       } catch (SQLException ex) {
         if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")) {
           System.out.println("Courses already exists !!!");
         } else {
           System.out.println("Courses : SQLException :
"+ex.getMessage());
         }
       }


     // Generating Table Colleges

       try {
         stmt.executeUpdate("create table or8regdfu.Colleges (college
VARCHAR(32), creqtable VARCHAR(32), primary key(college))");
         System.out.println("Colleges Created");
       } catch (SQLException ex) {
         if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")) {
           System.out.println("Colleges already exists !!!");
         } else {
           System.out.println("Colleges : SQLException :
"+ex.getMessage());
         }
       }


      // Generating Table CollegeRequirementsTables

       try {
         stmt.executeUpdate("create table
or8regdfu.CollegeRequirementsTables (creq VARCHAR(32), primary
key(creq) )");
         System.out.println("CollegeRequirementsTables Created");
       } catch (SQLException ex) {
         if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")) {
```

```
          System.out.println("CollegeRequirementsTables already exists
!!!");
        } else {
          System.out.println("CollegeRequirementsTables : SQLException
: "+ex.getMessage());
        }

      }



// Generating Table RequirementsTable

      try {
        stmt.executeUpdate("create table or8regdfu.RequirementsTable
(requirement VARCHAR(32), primary key(requirement) )");
        System.out.println("RequirementsTable Created");
      } catch (SQLException ex) {
        if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")) {
          System.out.println("RequirementsTable already exists !!!");
        } else {
          System.out.println("RequirementsTable : SQLException :
"+ex.getMessage());
        }
     }



// Generating Table SequencesTable

      try {
        stmt.executeUpdate("create table or8regdfu.SequencesTable
(sequence VARCHAR(32), primary key(sequence) )");
        System.out.println("SequencesTable Created");
      } catch (SQLException ex) {
        if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")){
          System.out.println("SequencesTable already exists !!!");
        } else {
          System.out.println("SequencesTable : SQLException :
"+ex.getMessage());
        }
      }



// Generating Table PermittedSequencesTables

      try {
```

```
        stmt.executeUpdate("create table
or8regdfu.PermittedSequencesTables (perseqtable VARCHAR(32), primary
key(perseqtable) )");
        System.out.println("PermittedSequencesTables Created");
      } catch (SQLException ex) {
        if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")){
          System.out.println("PermittedSequencesTables already exists
!!!");
        } else {
          System.out.println("PermittedSequencesTables : SQLException :
"+ex.getMessage());
        }
      }


    // Generating Table SpecialInfo

      try {
        stmt.executeUpdate("create table or8regdfu.SpecialInfo
(tablename VARCHAR(32), information VARCHAR(2000), primary
key(tablename) )");
        System.out.println("SpecialInfo Created");
      } catch (SQLException ex) {
        if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")){
          System.out.println("SpecialInfo already exists !!!");
        } else {
          System.out.println("SpecialInfo : SQLException :
"+ex.getMessage());
        }
      }


     // Generating Table Advisors

      try {
        stmt.executeUpdate("create table or8regdfu.Advisors (advisorID
VARCHAR(15) not NULL, Password VARCHAR(32), Name_Last "+
          "VARCHAR(40) not NULL, Name_First VARCHAR(20) not NULL,
Middlename VARCHAR(20), SSN CHAR(11) , Department VARCHAR(32) "+
          "not NULL, Grade VARCHAR(15), primary key(advisorID))");
        System.out.println("Advisors Created");
      } catch (SQLException ex) {
        if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")){
          System.out.println("Advisors already exists !!!");
        } else {
          System.out.println("Advisors : SQLException :
"+ex.getMessage());
        }
      }
```

```
// Generating Table Administrators

      successful = false;
      try {
        stmt.executeUpdate("create table or8regdfu.Administrators
(adminID VARCHAR(15) not NULL, Password VARCHAR(32), "+
          "primary key(adminID))");
        System.out.println("Administrators Created");
        successful = true;
      } catch (SQLException ex) {
        if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")){
          System.out.println("Administrators already exists !!!");
        } else {
          System.out.println("Administrators : SQLException :
"+ex.getMessage());
        }
      }



// Generating Table Students

      try {
        stmt.executeUpdate("create table or8regdfu.Students (netID
VARCHAR(15) not NULL, Password VARCHAR(32) not NULL, Name_Last "+
          "VARCHAR(40) not NULL, Name_First VARCHAR(20) not NULL,
Middlename VARCHAR(20), SSN CHAR(11) not NULL, inYear INTEGER "+
          "not NULL, Department VARCHAR(32) not NULL,
AcademicSpecialtyCode CHAR(4) not NULL, advisorID VARCHAR(15) not NULL,
"+
          "newRequest CHAR(1), Grade VARCHAR(15), Quota VARCHAR(8),
thesisquarter CHAR(4), primary key(netID), foreign key(advisorID) "+
          "references or8regdfu.Advisors)");
        System.out.println("Students Created");
      } catch (SQLException ex) {
        if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")){
          System.out.println("Students already exists !!!");
        } else {
          System.out.println("Students : SQLException :
"+ex.getMessage());
        }
      }



// Generating Table OfficialCourseList
```

```
      try {

        stmt.executeUpdate("create table or8regdfu.OfficialCourseList
(netID VARCHAR(15) not NULL, prefix CHAR(4) not NULL, codenum CHAR(3)
not NULL, "+
          "credit INTEGER, status CHAR(2), requirement VARCHAR(32),
sequence VARCHAR(32), quarter CHAR(4), variablecredit CHAR(1), foreign
key(netID) "+
          "references or8regdfu.Students)");

        System.out.println("OfficialCourseList created.");
      } catch (SQLException ex) {
        if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")){
          System.out.println("OfficialCourseList already exists !!!");
        } else {
          System.out.println("OfficialCourseList : SQLException :
"+ex.getMessage());
        }
      }


     // Generating Table RequestedCourseList

      try {

        stmt.executeUpdate("create table or8regdfu.RequestedCourseList
(netID VARCHAR(15) not NULL, prefix CHAR(4) not NULL, codenum CHAR(3)
not NULL, "+
          "credit INTEGER, status CHAR(2), requirement VARCHAR(32),
sequence VARCHAR(32), quarter CHAR(4), variablecredit CHAR(1), foreign
key(netID) "+
          "references or8regdfu.Students)");

        System.out.println("RequestedCourseList created.");
      } catch (SQLException ex) {
        if(ex.getMessage().equals("ORA-00955: name is already used by
an existing object")){
          System.out.println("RequestedCourseList already exists !!!");
        } else {
          System.out.println("RequestedCourseList : SQLException :
"+ex.getMessage());
        }
      }

     System.out.println("\nDatabase Generation Complete.\n");

     if(!user.equals("programadminverified")) {
      try {

        String stat = "insert into or8regdfu.Administrators values ('";
        stat = stat.concat(user);
```

```
        stat = stat.concat("', '");
        stat = stat.concat(password);
        stat = stat.concat("')");
        stmt.executeUpdate(stat);

      } catch (SQLException ex) {
        if(ex.getMessage().indexOf("ORA-00001: unique constraint") != -
1) {
          System.out.println("You are already registered as an
administrator.\n\n");
        } else if(ex.getMessage().equals("ORA-00942: table or view does
not exist") ) {
          System.out.println("The table Administrators does not exist.
There has been an error which must be "+
          "corrected before you can continue setup.  Either try running
this program again, or enter the database "+
          "and create the tables manually.  Make sure all tables are
under the or8regdfu schema.  You can find "+
          "the specs for the tables in the file dbdft.jag which can be
opened as normal text.\n\n");
        } else {
          System.out.println("\n\nThere was an error creating
Administrators or registering you as an administrator.  You may need to
run"+
          " this again or enter an SQL interface to manipulate the
database manually.  If you opt for the second, you must make sure "+
          "that table Administrators is created under the or8regdfu
schema with columns adminID VARCHAR(15) and Password VARCHAR(32), "+
          "and you must insert yourself into the administrators
table.\n\n");
        }
      }



      if(successful) {
        System.out.println("\nDatabase generation finished.\n\nYou are
now registered as an administrator for these programs.");
        System.out.println("As the initial administrator, you are
responsible for registering the other administrators and the"+
          "advisors.  You can do that through the Database
Administrative Tools GUI interface or manually through SQL.\n\n");
      } else {

      }

    }
  }
}
```

Appendix C. Schoolhouse OO Translation without Association Filtering

This AWSOME syntax was generated by the relational to OO schema translation application implemented in this research. The translation application was applied to the Schoolhouse relational schema seen in Appendix A prior to implementation of automated association filtering.

```
//*---It was POOR.......---*
//*---Now it's AWSOME !!!---*

//*---Types---*

type zeroToN is range 0 to 100000;
type zeroTo1 is range 0 to 1;
type oneToN is range 1 to 100000;
type oneTo1 is range 1 to 1;
type Integer is range -100000 to 100000;
type Date is array [1..9] of Char;
type StringSet is Set of String;

//*---Sets---*

type BICYCLESet is Set of BICYCLE;
type COURSESet is Set of COURSE;
type FACSet is Set of FAC;
type GRADCLASSSet is Set of GRADCLASS;
type PERSONSet is Set of PERSON;
type QUARSet is Set of QUAR;
type STUDSet is Set of STUD;
type SECTSet is Set of SECT;
type ASSIGNEDAOSet is Set of ASSIGNEDAO;
type OFFRNGSet is Set of OFFRNG;

//*---Basic Classes---*

Class BICYCLE is
  BIKESN :  String;
  OWNS__PERSON_SSN :  String;
end Class;
```

```
Class COURSE is
  CTYPE :  String;
  CNUM :  Integer;
  CTITLE :  String;
  CDESC :  String;
  CREDITHOURS :  Integer;
  LECTUREHOURS :  Integer;
  ABETDES :  Integer;
  ABETSCI :  Integer;
  ABETMATH :  Integer;
  ABETOTHER :  Integer;
end Class;

Class GRADCLASS is
  PROGRAM :  String;
  YEAR :  Integer;
  GRADDATE :  Date;
  DESIG :  String;
end Class;

Class PERSON is
  LASTNAME :  String;
  MIDINITIAL :  String;
  FIRSTNAME :  String;
  BIRTHDATE :  Date;
  SSN :  String;
  HEIGHT :  Integer;
  WEIGHT :  Integer;
end Class;

Class QUAR is
  QNAME :  String;
  YEAR :  Integer;
  QSTART :  Date;
  QEND :  Date;
end Class;

Class ASSIGNEDAO is
  STUD_PERSON_SSN :  String;
  SECT_TAS__OFFRNG_COURSE_CTITLE :  String;
  SECT_TAS__OFFRNG_QUAR_QNAME :  String;
end Class;

Class OFFRNG is
  CODE :  String;
  COURSE_CTITLE :  String;
  QUAR_QNAME :  String;
end Class;
```

```
//*---Derived Classes---*

Class FAC is PERSON with
  ACADEMICRANK :  String;
end Class;

Class STUD is PERSON with
  GPA :  Integer;
  MEMBEROF__GRADCLASS_DESIG :  String;
  RADVISES__FAC_PERSON_SSN :  String;
end Class;

Class SECT is OFFRNG with
  SNUMBER :  Integer;
  TEACHING__FAC_PERSON_SSN :  String;
end Class;

//*---Aggregation---*

Association OFFRNG_COURSE is
  role OFFRNG_COURSE_CTITLE-->COURSE_CTITLE : OFFRNG is oneToN;
  role COURSE_CTITLE-->OFFRNG_COURSE_CTITLE : COURSE is oneTo1;
end Association;

Association SECT_FAC is
  role SECT_TEACHING__FAC_PERSON_SSN-->FAC_PERSON_SSN : SECT is
zeroToN;
  role FAC_PERSON_SSN-->SECT_TEACHING__FAC_PERSON_SSN : FAC is oneTo1;
end Association;

Association STUD_FAC is
  role STUD_RADVISES__FAC_PERSON_SSN-->FAC_PERSON_SSN : STUD is
zeroToN;
  role FAC_PERSON_SSN-->STUD_RADVISES__FAC_PERSON_SSN : FAC is oneTo1;
end Association;

Association STUD_GRADCLASS is
  role STUD_MEMBEROF__GRADCLASS_DESIG-->GRADCLASS_DESIG : STUD is
zeroToN;
  role GRADCLASS_DESIG-->STUD_MEMBEROF__GRADCLASS_DESIG : GRADCLASS is
oneTo1;
end Association;

Association STUD_PERSON is
  role STUD_PERSON_SSN-->PERSON_SSN : STUD is oneToN;
  role PERSON_SSN-->STUD_PERSON_SSN : PERSON is oneTo1;
end Association;

Association OFFRNG_QUAR is
  role OFFRNG_QUAR_QNAME-->QUAR_QNAME : OFFRNG is oneToN;
  role QUAR_QNAME-->OFFRNG_QUAR_QNAME : QUAR is oneTo1;
end Association;
```

```
Association ASSIGNEDAO_STUD is
  role ASSIGNEDAO_STUD_PERSON_SSN-->STUD_PERSON_SSN : ASSIGNEDAO is
oneToN;
  role STUD_PERSON_SSN-->ASSIGNEDAO_STUD_PERSON_SSN : STUD is oneTo1;
end Association;

Association ASSIGNEDAO_SECT is
  role ASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTITLE--
>SECT_TAS__OFFRNG_COURSE_CTITLE : ASSIGNEDAO is oneToN;
  role SECT_TAS__OFFRNG_COURSE_CTITLE--
>ASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTITLE : SECT is oneTo1;
end Association;

Association SECT_OFFRNG is
  role SECT_TAS__OFFRNG_COURSE_CTITLE-->OFFRNG_COURSE_CTITLE : SECT is
oneToN;
  role OFFRNG_COURSE_CTITLE-->SECT_TAS__OFFRNG_COURSE_CTITLE : OFFRNG
is oneTo1;
end Association;

//*---Interactions---*

Association BICYCLE_PERSON is
  role BICYCLE_OWNS__PERSON_SSN-->PERSON_SSN : BICYCLE is zeroToN;
  role PERSON_SSN-->BICYCLE_OWNS__PERSON_SSN : PERSON is zeroTo1;
end Association;

Association FAC_PERSON is
  role FAC_PERSON_SSN-->PERSON_SSN : FAC is zeroTo1;
  role PERSON_SSN-->FAC_PERSON_SSN : PERSON is zeroTo1;
end Association;

//*---END OF OUTPUT---*
```

Appendix D. Schoolhouse OO Translation with Association Filtering

This AWSOME syntax was generated by the relational to OO schema translation application implemented in this research. The translation application was applied to the Schoolhouse relational schema seen in Appendix A after automated association filtering was implemented.

```
//Schoolhouse output in AWSOME syntax  with Association Filtering

//*---It was POOR.........---*
//*---Now it's AWSOME !!!---*

package ooSchema is

//*---Types---*

type zeroToN is range 0 .. 100000;
type zeroTo1 is range 0 .. 1;
type oneToN is range 1 .. 100000;
type oneTo1 is range 1 .. 1;
type Integer is range -100000 .. 100000;
type dateRange is range 1 .. 9;
type Date is array [dateRange] of Char;
type StringSet is Set of String;

//*---Sets---*

type BICYCLESet is set of BICYCLE;
type COURSESet is set of COURSE;
type FACSet is set of FAC;
type GRADCLASSSet is set of GRADCLASS;
type PERSONSet is set of PERSON;
type QUARSet is set of QUAR;
type STUDSet is set of STUD;
type SECTSet is set of SECT;
type ASSIGNEDAOSet is set of ASSIGNEDAO;
type OFFRNGSet is set of OFFRNG;
```

```
//*---Basic Classes---*

class BICYCLE is
  private BIKESN :  String;
  private OWNS__PERSON_SSN :  String;
end class;

class COURSE is
  private CTYPE :  String;
  private CNUM :  Integer;
  private CTITLE :  String;
  private CDESC :  String;
  private CREDITHOURS :  Integer;
  private LECTUREHOURS :  Integer;
  private ABETDES :  Integer;
  private ABETSCI :  Integer;
  private ABETMATH :  Integer;
  private ABETOTHER :  Integer;
end class;

class GRADCLASS is
  private PROGRAM :  String;
  private YEAR :  Integer;
  private GRADDATE :  Date;
  private DESIG :  String;
end class;

class PERSON is
  private LASTNAME :  String;
  private MIDINITIAL :  String;
  private FIRSTNAME :  String;
  private BIRTHDATE :  Date;
  private SSN :  String;
  private HEIGHT :  Integer;
  private WEIGHT :  Integer;
end class;

class QUAR is
  private QNAME :  String;
  private YEAR :  Integer;
  private QSTART :  Date;
  private QEND :  Date;
end class;

class ASSIGNEDAO is
  private STUD_PERSON_SSN :  String;
  private SECT_TAS__OFFRNG_COURSE_CTITLE :  String;
  private SECT_TAS__OFFRNG_QUAR_QNAME :  String;
end class;
```

```
class OFFRNG is
  private CODE :  String;
  private COURSE_CTITLE :  String;
  private QUAR_QNAME :  String;
end class;

//*---Derived Classes---*

class FAC is PERSON with
  private ACADEMICRANK :  String;
end class;

class STUD is PERSON with
  private GPA :  Integer;
  private MEMBEROF__GRADCLASS_DESIG :  String;
  private RADVISES__FAC_PERSON_SSN :  String;
end class;

class SECT is OFFRNG with
  private SNUMBER :  Integer;
  private TEACHING__FAC_PERSON_SSN :  String;
end class;

//*---Aggregation---*

association OFFRNG_COURSE is
  role OFFRNG_COURSE_CTITLEtoCOURSE_CTITLE : OFFRNG multiplicity
zeroTo1;
  role COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : COURSE multiplicity
zeroTo1;
end association;

association SECT_FAC is
  role SECT_TEACHING__FAC_PERSON_SSNtoFAC_PERSON_SSN : SECT
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSECT_TEACHING__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_FAC is
  role STUD_RADVISES__FAC_PERSON_SSNtoFAC_PERSON_SSN : STUD
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSTUD_RADVISES__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_GRADCLASS is
  role STUD_MEMBEROF__GRADCLASS_DESIGtoGRADCLASS_DESIG : STUD
multiplicity zeroToN;
  role GRADCLASS_DESIGtoSTUD_MEMBEROF__GRADCLASS_DESIG : GRADCLASS
multiplicity zeroTo1;
end association;
```

```
association STUD_PERSON is
  role STUD_PERSON_SSNtoPERSON_SSN : STUD multiplicity zeroTo1;
  role PERSON_SSNtoSTUD_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;

association OFFRNG_QUAR is
  role OFFRNG_QUAR_QNAMEtoQUAR_QNAME : OFFRNG multiplicity oneTo1;
  role QUAR_QNAMEtoOFFRNG_QUAR_QNAME : QUAR multiplicity zeroToN;
end association;

association ASSIGNEDAO_STUD is
  role ASSIGNEDAO_STUD_PERSON_SSNtoSTUD_PERSON_SSN : ASSIGNEDAO
multiplicity zeroTo1;
  role STUD_PERSON_SSNtoASSIGNEDAO_STUD_PERSON_SSN : STUD multiplicity
zeroTo1;
end association;


association ASSIGNEDAO_SECT is
  role
ASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTIT
LE : ASSIGNEDAO multiplicity oneTo1;
  role
SECT_TAS__OFFRNG_COURSE_CTITLEtoASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTIT
LE : SECT multiplicity oneTo1;
end association;

association SECT_OFFRNG is
  role SECT_TAS__OFFRNG_COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : SECT
multiplicity zeroTo1;
  role OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTITLE : OFFRNG
multiplicity zeroTo1;
end association;

//*---Interactions---*

association BICYCLE_PERSON is
  role BICYCLE_OWNS__PERSON_SSNtoPERSON_SSN : BICYCLE multiplicity
zeroToN;
  role PERSON_SSNtoBICYCLE_OWNS__PERSON_SSN : PERSON multiplicity
zeroTo1;
end association;

association FAC_PERSON is
  role FAC_PERSON_SSNtoPERSON_SSN : FAC multiplicity zeroTo1;
  role PERSON_SSNtoFAC_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;
end package;

//*---END OF OUTPUT---*
```

Appendix E. Validation of Schema Change Detection and Translation Regeneration

This Appendix shows the validation protocol and results for the *tunable-temporal* Schema Change Detection and Regeneration scheme when applied to the schoolhouse schema. The SQL commands that modify the schema and the resulting change to the re-generated AWSOME syntax are in **bold**. When the schema change causes a removal from the AWSOME output, "***//REMOVED***" is shown where the output was prior to removal. The specific schema changes tested are:

- Relation added to the schema
- Relation removed from the schema
- Column added to a relation
- Column data type modified
- Association multiplicity (cardinality) change

```
//-----------------------------------------------------------------------
SQL>  create table tutor(
  2   name         varchar2(40) not null,
  3  when        varchar2(9),
  4   primary key (name));

SQL> create table seminar(
  2  topic       varchar2(40) not null,
  3   when        varchar2(9),
  4  primary key (topic));

SQL> describe tutor;
```

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 NAME                                      NOT NULL VARCHAR2(40)
 WHEN                                               VARCHAR2(9)

SQL> describe seminar;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 TOPIC                                     NOT NULL VARCHAR2(40)
 WHEN                                               VARCHAR2(9)

//*---It was POOR.........---*
//*---Now it's AWSOME !!!---*

package ooSchema is

//*---Types---*

type zeroToN is range 0 .. 100000;
type zeroTo1 is range 0 .. 1;
type oneToN is range 1 .. 100000;
type oneTo1 is range 1 .. 1;
type Integer is range -100000 .. 100000;
type dateRange is range 1 .. 9;
type Date is array [dateRange] of Char;
type StringSet is Set of String;

//*---Sets---*

type BICYCLESet is set of BICYCLE;
type COURSESet is set of COURSE;
type FACSet is set of FAC;
type GRADCLASSSet is set of GRADCLASS;
type PERSONSet is set of PERSON;
type QUARSet is set of QUAR;
type SEMINARSet is set of SEMINAR;
type STUDSet is set of STUD;
type TUTORSet is set of TUTOR;
type SECTSet is set of SECT;
type ASSIGNEDAOSet is set of ASSIGNEDAO;
type OFFRNGSet is set of OFFRNG;


//*---Basic Classes---*

class BICYCLE is
  private BIKESN :  String;
  private OWNS__PERSON_SSN :  String;
end class;

class COURSE is
  private CTYPE :  String;
  private CNUM :  Integer;
  private CTITLE :  String;
```

```
  private CDESC :  String;
  private CREDITHOURS :  Integer;
  private LECTUREHOURS :  Integer;
  private ABETDES :  Integer;
  private ABETSCI :  Integer;
  private ABETMATH :  Integer;
  private ABETOTHER :  Integer;
end class;

class GRADCLASS is
  private PROGRAM :  String;
  private YEAR :  Integer;
  private GRADDATE :  Date;
  private DESIG :  String;
end class;

class PERSON is
  private LASTNAME :  String;
  private MIDINITIAL :  String;
  private FIRSTNAME :  String;
  private BIRTHDATE :  Date;
  private SSN :  String;
  private HEIGHT :  Integer;
  private WEIGHT :  Integer;
end class;

class QUAR is
  private QNAME :  String;
  private YEAR :  Integer;
  private QSTART :  Date;
  private QEND :  Date;
end class;

class SEMINAR is
  private TOPIC :  String;
  private WHEN :  String;
end class;

class TUTOR is
  private NAME :  String;
  private WHEN :  String;
end class;

class ASSIGNEDAO is
  private STUD_PERSON_SSN :  String;
  private SECT_TAS__OFFRNG_COURSE_CTITLE :  String;
  private SECT_TAS__OFFRNG_QUAR_QNAME :  String;
end class;

class OFFRNG is
  private CODE :  String;
  private COURSE_CTITLE :  String;
  private QUAR_QNAME :  String;
end class;
```

```
//*---Derived Classes---*

class FAC is PERSON with
  private ACADEMICRANK :  String;
end class;

class STUD is PERSON with
  private GPA :  Integer;
  private MEMBEROF__GRADCLASS_DESIG :  String;
  private RADVISES__FAC_PERSON_SSN :  String;
  private SAT_SCORE :  String;
  private IQ :  Integer;
end class;

class SECT is OFFRNG with
  private SNUMBER :  Integer;
  private TEACHING__FAC_PERSON_SSN :  String;
end class;

//*---Aggregation---*

association OFFRNG_COURSE is
  role OFFRNG_COURSE_CTITLEtoCOURSE_CTITLE : OFFRNG multiplicity
zeroTo1;
  role COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : COURSE multiplicity
zeroTo1;
end association;

association SECT_FAC is
  role SECT_TEACHING__FAC_PERSON_SSNtoFAC_PERSON_SSN : SECT
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSECT_TEACHING__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_FAC is
  role STUD_RADVISES__FAC_PERSON_SSNtoFAC_PERSON_SSN : STUD
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSTUD_RADVISES__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_GRADCLASS is
  role STUD_MEMBEROF__GRADCLASS_DESIGtoGRADCLASS_DESIG : STUD
multiplicity zeroToN;
  role GRADCLASS_DESIGtoSTUD_MEMBEROF__GRADCLASS_DESIG : GRADCLASS
multiplicity zeroTo1;
end association;

association STUD_PERSON is
  role STUD_PERSON_SSNtoPERSON_SSN : STUD multiplicity zeroTo1;
  role PERSON_SSNtoSTUD_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;
```

```
association OFFRNG_QUAR is
  role OFFRNG_QUAR_QNAMEtoQUAR_QNAME : OFFRNG multiplicity oneTo1;
  role QUAR_QNAMEtoOFFRNG_QUAR_QNAME : QUAR multiplicity zeroToN;
end association;

association ASSIGNEDAO_STUD is
  role ASSIGNEDAO_STUD_PERSON_SSNtoSTUD_PERSON_SSN : ASSIGNEDAO
multiplicity zeroTo1;
  role STUD_PERSON_SSNtoASSIGNEDAO_STUD_PERSON_SSN : STUD multiplicity
zeroTo1;
end association;

association ASSIGNEDAO_SECT is
  role
ASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTIT
LE : ASSIGNEDAO multiplicity oneTo1;
  role
SECT_TAS__OFFRNG_COURSE_CTITLEtoASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTIT
LE : SECT multiplicity oneTo1;
end association;

association SECT_OFFRNG is
  role SECT_TAS__OFFRNG_COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : SECT
multiplicity zeroTo1;
  role OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTITLE : OFFRNG
multiplicity zeroTo1;
end association;

//*---Interactions---*

association BICYCLE_PERSON is
  role BICYCLE_OWNS__PERSON_SSNtoPERSON_SSN : BICYCLE multiplicity
zeroToN;
  role PERSON_SSNtoBICYCLE_OWNS__PERSON_SSN : PERSON multiplicity
zeroTo1;
end association;

association FAC_PERSON is
  role FAC_PERSON_SSNtoPERSON_SSN : FAC multiplicity zeroTo1;
  role PERSON_SSNtoFAC_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;
end package;

//*---END OF OUTPUT---*

//-------------------------------------------------------------------------
SQL> drop table seminar;

//*---It was POOR.....---*
//*---Now it's AWSOME !!!---*

package ooSchema is
```

```
//*---Types---*

type zeroToN is range 0 .. 100000;
type zeroTo1 is range 0 .. 1;
type oneToN is range 1 .. 100000;
type oneTo1 is range 1 .. 1;
type Integer is range -100000 .. 100000;
type dateRange is range 1 .. 9;
type Date is array [dateRange] of Char;
type StringSet is Set of String;

//*---Sets---*

type BICYCLESet is set of BICYCLE;
type COURSESet is set of COURSE;
type FACSet is set of FAC;
type GRADCLASSSet is set of GRADCLASS;
type PERSONSet is set of PERSON;
type QUARSet is set of QUAR;
type STUDSet is set of STUD;
type TUTORSet is set of TUTOR;
type SECTSet is set of SECT;
type ASSIGNEDAOSet is set of ASSIGNEDAO;
type OFFRNGSet is set of OFFRNG;


//*---Basic Classes---*

class BICYCLE is
  private BIKESN :  String;
  private OWNS__PERSON_SSN :  String;
end class;

class COURSE is
  private CTYPE :  String;
  private CNUM :  Integer;
  private CTITLE :  String;
  private CDESC :  String;
  private CREDITHOURS :  Integer;
  private LECTUREHOURS :  Integer;
  private ABETDES :  Integer;
  private ABETSCI :  Integer;
  private ABETMATH :  Integer;
  private ABETOTHER :  Integer;
end class;

class GRADCLASS is
  private PROGRAM :  String;
  private YEAR :  Integer;
  private GRADDATE :  Date;
  private DESIG :  String;
end class;

class PERSON is
```

```
  private LASTNAME :  String;
  private MIDINITIAL :  String;
  private FIRSTNAME :  String;
  private BIRTHDATE :  Date;
  private SSN :  String;
  private HEIGHT :  Integer;
  private WEIGHT :  Integer;
end class;

class QUAR is
  private QNAME :  String;
  private YEAR :  Integer;
  private QSTART :  Date;
  private QEND :  Date;
end class;

//REMOVED

class TUTOR is
  private NAME :  String;
  private WHEN :  String;
end class;

class ASSIGNEDAO is
  private STUD_PERSON_SSN :  String;
  private SECT_TAS__OFFRNG_COURSE_CTITLE :  String;
  private SECT_TAS__OFFRNG_QUAR_QNAME :  String;
end class;

class OFFRNG is
  private CODE :  String;
  private COURSE_CTITLE :  String;
  private QUAR_QNAME :  String;
end class;

//*---Derived Classes---*

class FAC is PERSON with
  private ACADEMICRANK :  String;
end class;

class STUD is PERSON with
  private GPA :  Integer;
  private MEMBEROF__GRADCLASS_DESIG :  String;
  private RADVISES__FAC_PERSON_SSN :  String;
  private SAT_SCORE :  String;
  private IQ :  Integer;
end class;

class SECT is OFFRNG with
  private SNUMBER :  Integer;
  private TEACHING__FAC_PERSON_SSN :  String;
end class;
```

```
//*---Aggregation---*

association OFFRNG_COURSE is
  role OFFRNG_COURSE_CTITLEtoCOURSE_CTITLE : OFFRNG multiplicity
zeroTo1;
  role COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : COURSE multiplicity
zeroTo1;
end association;

association SECT_FAC is
  role SECT_TEACHING__FAC_PERSON_SSNtoFAC_PERSON_SSN : SECT
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSECT_TEACHING__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_FAC is
  role STUD_RADVISES__FAC_PERSON_SSNtoFAC_PERSON_SSN : STUD
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSTUD_RADVISES__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_GRADCLASS is
  role STUD_MEMBEROF__GRADCLASS_DESIGtoGRADCLASS_DESIG : STUD
multiplicity zeroToN;
  role GRADCLASS_DESIGtoSTUD_MEMBEROF__GRADCLASS_DESIG : GRADCLASS
multiplicity zeroTo1;
end association;

association STUD_PERSON is
  role STUD_PERSON_SSNtoPERSON_SSN : STUD multiplicity zeroTo1;
  role PERSON_SSNtoSTUD_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;

association OFFRNG_QUAR is
  role OFFRNG_QUAR_QNAMEtoQUAR_QNAME : OFFRNG multiplicity oneTo1;
  role QUAR_QNAMEtoOFFRNG_QUAR_QNAME : QUAR multiplicity zeroToN;
end association;

association ASSIGNEDAO_STUD is
  role ASSIGNEDAO_STUD_PERSON_SSNtoSTUD_PERSON_SSN : ASSIGNEDAO
multiplicity zeroTo1;
  role STUD_PERSON_SSNtoASSIGNEDAO_STUD_PERSON_SSN : STUD multiplicity
zeroTo1;
end association;

association ASSIGNEDAO_SECT is
  role
ASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTIT
LE : ASSIGNEDAO multiplicity oneTo1;
  role
SECT_TAS__OFFRNG_COURSE_CTITLEtoASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTIT
LE : SECT multiplicity oneTo1;
```

```
end association;

association SECT_OFFRNG is
  role SECT_TAS__OFFRNG_COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : SECT
multiplicity zeroTo1;
  role OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTITLE : OFFRNG
multiplicity zeroTo1;
end association;

//*---Interactions---*

association BICYCLE_PERSON is
  role BICYCLE_OWNS__PERSON_SSNtoPERSON_SSN : BICYCLE multiplicity
zeroToN;
  role PERSON_SSNtoBICYCLE_OWNS__PERSON_SSN : PERSON multiplicity
zeroTo1;
end association;

association FAC_PERSON is
  role FAC_PERSON_SSNtoPERSON_SSN : FAC multiplicity zeroTo1;
  role PERSON_SSNtoFAC_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;
end package;

//*---END OF OUTPUT---*

//-------------------------------------------------------------------------
SQL> create table seminar(
  2  topic      varchar2(40) not null,
  3   when       varchar2(9),
  4   primary key (topic));

Table created.

//*---It was POOR.........---*
//*---Now it's AWSOME !!!---*

package ooSchema is

//*---Types---*

type zeroToN is range 0 .. 100000;
type zeroTo1 is range 0 .. 1;
type oneToN is range 1 .. 100000;
type oneTo1 is range 1 .. 1;
type Integer is range -100000 .. 100000;
type dateRange is range 1 .. 9;
type Date is array [dateRange] of Char;
type StringSet is Set of String;

//*---Sets---*

type BICYCLESet is set of BICYCLE;
type COURSESet is set of COURSE;
```

```
type FACSet is set of FAC;
type GRADCLASSSet is set of GRADCLASS;
type PERSONSet is set of PERSON;
type QUARSet is set of QUAR;
type SEMINARSet is set of SEMINAR;
type STUDSet is set of STUD;
type TUTORSet is set of TUTOR;
type SECTSet is set of SECT;
type ASSIGNEDAOSet is set of ASSIGNEDAO;
type OFFRNGSet is set of OFFRNG;


//*---Basic Classes---*

class BICYCLE is
  private BIKESN :  String;
  private OWNS__PERSON_SSN :  String;
end class;

class COURSE is
  private CTYPE :  String;
  private CNUM :  Integer;
  private CTITLE :  String;
  private CDESC :  String;
  private CREDITHOURS :  Integer;
  private LECTUREHOURS :  Integer;
  private ABETDES :  Integer;
  private ABETSCI :  Integer;
  private ABETMATH :  Integer;
  private ABETOTHER :  Integer;
end class;

class GRADCLASS is
  private PROGRAM :  String;
  private YEAR :  Integer;
  private GRADDATE :  Date;
  private DESIG :  String;
end class;

class PERSON is
  private LASTNAME :  String;
  private MIDINITIAL :  String;
  private FIRSTNAME :  String;
  private BIRTHDATE :  Date;
  private SSN :  String;
  private HEIGHT :  Integer;
  private WEIGHT :  Integer;
end class;

class QUAR is
  private QNAME :  String;
  private YEAR :  Integer;
  private QSTART :  Date;
  private QEND :  Date;
```

```
end class;

//-------------------------------------------------------------------------
class SEMINAR is
  private TOPIC :  String;
  private WHEN :  String;
end class;

class TUTOR is
  private NAME :  String;
  private WHEN :  String;
end class;

class ASSIGNEDAO is
  private STUD_PERSON_SSN :  String;
  private SECT_TAS__OFFRNG_COURSE_CTITLE :  String;
  private SECT_TAS__OFFRNG_QUAR_QNAME :  String;
end class;

class OFFRNG is
  private CODE :  String;
  private COURSE_CTITLE :  String;
  private QUAR_QNAME :  String;
end class;

//*---Derived Classes---*

class FAC is PERSON with
  private ACADEMICRANK :  String;
end class;

class STUD is PERSON with
  private GPA :  Integer;
  private MEMBEROF__GRADCLASS_DESIG :  String;
  private RADVISES__FAC_PERSON_SSN :  String;
  private SAT_SCORE :  String;
  private IQ :  Integer;
end class;

class SECT is OFFRNG with
  private SNUMBER :  Integer;
  private TEACHING__FAC_PERSON_SSN :  String;
end class;

//*---Aggregation---*

association OFFRNG_COURSE is
  role OFFRNG_COURSE_CTITLEtoCOURSE_CTITLE : OFFRNG multiplicity
zeroTo1;
  role COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : COURSE multiplicity
zeroTo1;
end association;

association SECT_FAC is
```

```
  role SECT_TEACHING__FAC_PERSON_SSNtoFAC_PERSON_SSN : SECT
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSECT_TEACHING__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_FAC is
  role STUD_RADVISES__FAC_PERSON_SSNtoFAC_PERSON_SSN : STUD
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSTUD_RADVISES__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_GRADCLASS is
  role STUD_MEMBEROF__GRADCLASS_DESIGtoGRADCLASS_DESIG : STUD
multiplicity zeroToN;
  role GRADCLASS_DESIGtoSTUD_MEMBEROF__GRADCLASS_DESIG : GRADCLASS
multiplicity zeroTo1;
end association;

association STUD_PERSON is
  role STUD_PERSON_SSNtoPERSON_SSN : STUD multiplicity zeroTo1;
  role PERSON_SSNtoSTUD_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;

association OFFRNG_QUAR is
  role OFFRNG_QUAR_QNAMEtoQUAR_QNAME : OFFRNG multiplicity oneTo1;
  role QUAR_QNAMEtoOFFRNG_QUAR_QNAME : QUAR multiplicity zeroToN;
end association;

association ASSIGNEDAO_STUD is
  role ASSIGNEDAO_STUD_PERSON_SSNtoSTUD_PERSON_SSN : ASSIGNEDAO
multiplicity zeroTo1;
  role STUD_PERSON_SSNtoASSIGNEDAO_STUD_PERSON_SSN : STUD multiplicity
zeroTo1;
end association;

association ASSIGNEDAO_SECT is
  role
ASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTIT
LE : ASSIGNEDAO multiplicity oneTo1;
  role
SECT_TAS__OFFRNG_COURSE_CTITLEtoASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTIT
LE : SECT multiplicity oneTo1;
end association;

association SECT_OFFRNG is
  role SECT_TAS__OFFRNG_COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : SECT
multiplicity zeroTo1;
  role OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTITLE : OFFRNG
multiplicity zeroTo1;
end association;

//*---Interactions---*
```

```
association BICYCLE_PERSON is
  role BICYCLE_OWNS__PERSON_SSNtoPERSON_SSN : BICYCLE multiplicity
zeroToN;
  role PERSON_SSNtoBICYCLE_OWNS__PERSON_SSN : PERSON multiplicity
zeroTo1;
end association;

association FAC_PERSON is
  role FAC_PERSON_SSNtoPERSON_SSN : FAC multiplicity zeroTo1;
  role PERSON_SSNtoFAC_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;
end package;

//*---END OF OUTPUT---*

//-----------------------------------------------------------------------
SQL> alter table seminar add (cost integer);

Table altered.

//*---It was POOR.........---*
//*---Now it's AWSOME !!!---*

package ooSchema is

//*---Types---*

type zeroToN is range 0 .. 100000;
type zeroTo1 is range 0 .. 1;
type oneToN is range 1 .. 100000;
type oneTo1 is range 1 .. 1;
type Integer is range -100000 .. 100000;
type dateRange is range 1 .. 9;
type Date is array [dateRange] of Char;
type StringSet is Set of String;

//*---Sets---*

type BICYCLESet is set of BICYCLE;
type COURSESet is set of COURSE;
type FACSet is set of FAC;
type GRADCLASSSet is set of GRADCLASS;
type PERSONSet is set of PERSON;
type QUARSet is set of QUAR;
type SEMINARSet is set of SEMINAR;
type STUDSet is set of STUD;
type TUTORSet is set of TUTOR;
type SECTSet is set of SECT;
type ASSIGNEDAOSet is set of ASSIGNEDAO;
type OFFRNGSet is set of OFFRNG;


//*---Basic Classes---*
```

```
class BICYCLE is
  private BIKESN :  String;
  private OWNS__PERSON_SSN :  String;
end class;

class COURSE is
  private CTYPE :  String;
  private CNUM :  Integer;
  private CTITLE :  String;
  private CDESC :  String;
  private CREDITHOURS :  Integer;
  private LECTUREHOURS :  Integer;
  private ABETDES :  Integer;
  private ABETSCI :  Integer;
  private ABETMATH :  Integer;
  private ABETOTHER :  Integer;
end class;

class GRADCLASS is
  private PROGRAM :  String;
  private YEAR :  Integer;
  private GRADDATE :  Date;
  private DESIG :  String;
end class;

class PERSON is
  private LASTNAME :  String;
  private MIDINITIAL :  String;
  private FIRSTNAME :  String;
  private BIRTHDATE :  Date;
  private SSN :  String;
  private HEIGHT :  Integer;
  private WEIGHT :  Integer;
end class;

class QUAR is
  private QNAME :  String;
  private YEAR :  Integer;
  private QSTART :  Date;
  private QEND :  Date;
end class;

class SEMINAR is
  private TOPIC :  String;
  private WHEN :  String;
  private COST :  Integer;
end class;

class TUTOR is
  private NAME :  String;
  private WHEN :  String;
end class;
```

```
class ASSIGNEDAO is
  private STUD_PERSON_SSN :  String;
  private SECT_TAS__OFFRNG_COURSE_CTITLE :  String;
  private SECT_TAS__OFFRNG_QUAR_QNAME :  String;
end class;

class OFFRNG is
  private CODE :  String;
  private COURSE_CTITLE :  String;
  private QUAR_QNAME :  String;
end class;

//*---Derived Classes---*

class FAC is PERSON with
  private ACADEMICRANK :  String;
end class;

class STUD is PERSON with
  private GPA :  Integer;
  private MEMBEROF__GRADCLASS_DESIG :  String;
  private RADVISES__FAC_PERSON_SSN :  String;
  private SAT_SCORE :  String;
  private IQ :  Integer;
end class;

class SECT is OFFRNG with
  private SNUMBER :  Integer;
  private TEACHING__FAC_PERSON_SSN :  String;
end class;

//*---Aggregation---*

association OFFRNG_COURSE is
  role OFFRNG_COURSE_CTITLEtoCOURSE_CTITLE : OFFRNG multiplicity
zeroTo1;
  role COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : COURSE multiplicity
zeroTo1;
end association;

association SECT_FAC is
  role SECT_TEACHING__FAC_PERSON_SSNtoFAC_PERSON_SSN : SECT
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSECT_TEACHING__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_FAC is
  role STUD_RADVISES__FAC_PERSON_SSNtoFAC_PERSON_SSN : STUD
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSTUD_RADVISES__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;
```

```
association STUD_GRADCLASS is
  role STUD_MEMBEROF__GRADCLASS_DESIGtoGRADCLASS_DESIG : STUD
multiplicity zeroToN;
  role GRADCLASS_DESIGtoSTUD_MEMBEROF__GRADCLASS_DESIG : GRADCLASS
multiplicity zeroTo1;
end association;

association STUD_PERSON is
  role STUD_PERSON_SSNtoPERSON_SSN : STUD multiplicity zeroTo1;
  role PERSON_SSNtoSTUD_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;

association OFFRNG_QUAR is
  role OFFRNG_QUAR_QNAMEtoQUAR_QNAME : OFFRNG multiplicity oneTo1;
  role QUAR_QNAMEtoOFFRNG_QUAR_QNAME : QUAR multiplicity zeroToN;
end association;

association ASSIGNEDAO_STUD is
  role ASSIGNEDAO_STUD_PERSON_SSNtoSTUD_PERSON_SSN : ASSIGNEDAO
multiplicity zeroTo1;
  role STUD_PERSON_SSNtoASSIGNEDAO_STUD_PERSON_SSN : STUD multiplicity
zeroTo1;
end association;

association ASSIGNEDAO_SECT is
  role
ASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTIT
LE : ASSIGNEDAO multiplicity oneTo1;
  role
SECT_TAS__OFFRNG_COURSE_CTITLEtoASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTIT
LE : SECT multiplicity oneTo1;
end association;

association SECT_OFFRNG is
  role SECT_TAS__OFFRNG_COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : SECT
multiplicity zeroTo1;
  role OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTITLE : OFFRNG
multiplicity zeroTo1;
end association;

//*---Interactions---*

association BICYCLE_PERSON is
  role BICYCLE_OWNS__PERSON_SSNtoPERSON_SSN : BICYCLE multiplicity
zeroToN;
  role PERSON_SSNtoBICYCLE_OWNS__PERSON_SSN : PERSON multiplicity
zeroTo1;
end association;

association FAC_PERSON is
  role FAC_PERSON_SSNtoPERSON_SSN : FAC multiplicity zeroTo1;
  role PERSON_SSNtoFAC_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;
end package;
```

```
//*---END OF OUTPUT---*

//-------------------------------------------------------------------------
SQL> alter table seminar modify(cost varchar2(9));


//*---It was POOR.....---*
//*---Now it's AWSOME !!!---*

package ooSchema is

//*---Types---*

type zeroToN is range 0 .. 100000;
type zeroTo1 is range 0 .. 1;
type oneToN is range 1 .. 100000;
type oneTo1 is range 1 .. 1;
type Integer is range -100000 .. 100000;
type dateRange is range 1 .. 9;
type Date is array [dateRange] of Char;
type StringSet is Set of String;

//*---Sets---*

type BICYCLESet is set of BICYCLE;
type COURSESet is set of COURSE;
type FACSet is set of FAC;
type GRADCLASSSet is set of GRADCLASS;
type PERSONSet is set of PERSON;
type QUARSet is set of QUAR;
type SEMINARSet is set of SEMINAR;
type STUDSet is set of STUD;
type TUTORSet is set of TUTOR;
type SECTSet is set of SECT;
type ASSIGNEDAOSet is set of ASSIGNEDAO;
type OFFRNGSet is set of OFFRNG;


//*---Basic Classes---*

class BICYCLE is
  private BIKESN :  String;
  private OWNS__PERSON_SSN :  String;
end class;

class COURSE is
  private CTYPE :  String;
  private CNUM :  Integer;
  private CTITLE :  String;
  private CDESC :  String;
  private CREDITHOURS :  Integer;
  private LECTUREHOURS :  Integer;
  private ABETDES :  Integer;
```

```
  private ABETSCI :  Integer;
  private ABETMATH :  Integer;
  private ABETOTHER :  Integer;
end class;

class GRADCLASS is
  private PROGRAM :  String;
  private YEAR :  Integer;
  private GRADDATE :  Date;
  private DESIG :  String;
end class;

class PERSON is
  private LASTNAME :  String;
  private MIDINITIAL :  String;
  private FIRSTNAME :  String;
  private BIRTHDATE :  Date;
  private SSN :  String;
  private HEIGHT :  Integer;
  private WEIGHT :  Integer;
end class;

class QUAR is
  private QNAME :  String;
  private YEAR :  Integer;
  private QSTART :  Date;
  private QEND :  Date;
end class;

class SEMINAR is
  private TOPIC :  String;
  private WHEN :  String;
  private COST :  String;
end class;

class TUTOR is
  private NAME :  String;
  private WHEN :  String;
end class;

class ASSIGNEDAO is
  private STUD_PERSON_SSN :  String;
  private SECT_TAS__OFFRNG_COURSE_CTITLE :  String;
  private SECT_TAS__OFFRNG_QUAR_QNAME :  String;
end class;

class OFFRNG is
  private CODE :  String;
  private COURSE_CTITLE :  String;
  private QUAR_QNAME :  String;
end class;

//*---Derived Classes---*
```

```
class FAC is PERSON with
  private ACADEMICRANK :  String;
end class;

class STUD is PERSON with
  private GPA :  Integer;
  private MEMBEROF__GRADCLASS_DESIG :  String;
  private RADVISES__FAC_PERSON_SSN :  String;
  private SAT_SCORE :  String;
  private IQ :  Integer;
end class;

class SECT is OFFRNG with
  private SNUMBER :  Integer;
  private TEACHING__FAC_PERSON_SSN :  String;
end class;

//*---Aggregation---*

association OFFRNG_COURSE is
  role OFFRNG_COURSE_CTITLEtoCOURSE_CTITLE : OFFRNG multiplicity
zeroTo1;
  role COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : COURSE multiplicity
zeroTo1;
end association;

association SECT_FAC is
  role SECT_TEACHING__FAC_PERSON_SSNtoFAC_PERSON_SSN : SECT
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSECT_TEACHING__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_FAC is
  role STUD_RADVISES__FAC_PERSON_SSNtoFAC_PERSON_SSN : STUD
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSTUD_RADVISES__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_GRADCLASS is
  role STUD_MEMBEROF__GRADCLASS_DESIGtoGRADCLASS_DESIG : STUD
multiplicity zeroToN;
  role GRADCLASS_DESIGtoSTUD_MEMBEROF__GRADCLASS_DESIG : GRADCLASS
multiplicity zeroTo1;
end association;

association STUD_PERSON is
  role STUD_PERSON_SSNtoPERSON_SSN : STUD multiplicity zeroTo1;
  role PERSON_SSNtoSTUD_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;

association OFFRNG_QUAR is
  role OFFRNG_QUAR_QNAMEtoQUAR_QNAME : OFFRNG multiplicity oneTo1;
```

```
  role QUAR_QNAMEtoOFFRNG_QUAR_QNAME : QUAR multiplicity zeroToN;
end association;

association ASSIGNEDAO_STUD is
  role ASSIGNEDAO_STUD_PERSON_SSNtoSTUD_PERSON_SSN : ASSIGNEDAO
multiplicity zeroTo1;
  role STUD_PERSON_SSNtoASSIGNEDAO_STUD_PERSON_SSN : STUD multiplicity
zeroTo1;
end association;

association ASSIGNEDAO_SECT is
  role
ASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTIT
LE : ASSIGNEDAO multiplicity oneTo1;
  role
SECT_TAS__OFFRNG_COURSE_CTITLEtoASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTIT
LE : SECT multiplicity oneTo1;
end association;

association SECT_OFFRNG is
  role SECT_TAS__OFFRNG_COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : SECT
multiplicity zeroTo1;
  role OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTITLE : OFFRNG
multiplicity zeroTo1;
end association;

//*---Interactions---*

association BICYCLE_PERSON is
  role BICYCLE_OWNS__PERSON_SSNtoPERSON_SSN : BICYCLE multiplicity
zeroToN;
  role PERSON_SSNtoBICYCLE_OWNS__PERSON_SSN : PERSON multiplicity
zeroTo1;
end association;

association FAC_PERSON is
  role FAC_PERSON_SSNtoPERSON_SSN : FAC multiplicity zeroTo1;
  role PERSON_SSNtoFAC_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;
end package;

//*---END OF OUTPUT---*

SQL>  delete from assignedao where  SECT_TAS__OFFRNG_QUAR_QNAME = 'Fall
00';

2 rows deleted.


//*---It was POOR.........---*
//*---Now it's AWSOME !!!---*

package ooSchema is
```

```
//*---Types---*

type zeroToN is range 0 .. 100000;
type zeroTo1 is range 0 .. 1;
type oneToN is range 1 .. 100000;
type oneTo1 is range 1 .. 1;
type Integer is range -100000 .. 100000;
type dateRange is range 1 .. 9;
type Date is array [dateRange] of Char;
type StringSet is Set of String;

//*---Sets---*

type BICYCLESet is set of BICYCLE;
type COURSESet is set of COURSE;
type FACSet is set of FAC;
type GRADCLASSSet is set of GRADCLASS;
type PERSONSet is set of PERSON;
type QUARSet is set of QUAR;
type SEMINARSet is set of SEMINAR;
type STUDSet is set of STUD;
type TUTORSet is set of TUTOR;
type SECTSet is set of SECT;
type ASSIGNEDAOSet is set of ASSIGNEDAO;
type OFFRNGSet is set of OFFRNG;


//*---Basic Classes---*

class BICYCLE is
  private BIKESN :  String;
  private OWNS__PERSON_SSN :  String;
end class;

class COURSE is
  private CTYPE :  String;
  private CNUM :  Integer;
  private CTITLE :  String;
  private CDESC :  String;
  private CREDITHOURS :  Integer;
  private LECTUREHOURS :  Integer;
  private ABETDES :  Integer;
  private ABETSCI :  Integer;
  private ABETMATH :  Integer;
  private ABETOTHER :  Integer;
end class;

class GRADCLASS is
  private PROGRAM :  String;
  private YEAR :  Integer;
  private GRADDATE :  Date;
  private DESIG :  String;
end class;
```

```
class PERSON is
  private LASTNAME :  String;
  private MIDINITIAL :  String;
  private FIRSTNAME :  String;
  private BIRTHDATE :  Date;
  private SSN :  String;
  private HEIGHT :  Integer;
  private WEIGHT :  Integer;
end class;

class QUAR is
  private QNAME :  String;
  private YEAR :  Integer;
  private QSTART :  Date;
  private QEND :  Date;
end class;

class SEMINAR is
  private TOPIC :  String;
  private WHEN :  String;
  private COST :  String;
end class;

class TUTOR is
  private NAME :  String;
  private WHEN :  String;
end class;

class ASSIGNEDAO is
  private STUD_PERSON_SSN :  String;
  private SECT_TAS__OFFRNG_COURSE_CTITLE :  String;
  private SECT_TAS__OFFRNG_QUAR_QNAME :  String;
end class;

class OFFRNG is
  private CODE :  String;
  private COURSE_CTITLE :  String;
  private QUAR_QNAME :  String;
end class;

//*---Derived Classes---*

class FAC is PERSON with
  private ACADEMICRANK :  String;
end class;

class STUD is PERSON with
  private GPA :  Integer;
  private MEMBEROF__GRADCLASS_DESIG :  String;
  private RADVISES__FAC_PERSON_SSN :  String;
  private SAT_SCORE :  String;
  private IQ :  Integer;
end class;
```

```
class SECT is OFFRNG with
  private SNUMBER :  Integer;
  private TEACHING__FAC_PERSON_SSN :  String;
end class;

//*---Aggregation---*

association OFFRNG_COURSE is
  role OFFRNG_COURSE_CTITLEtoCOURSE_CTITLE : OFFRNG multiplicity
zeroTo1;
  role COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : COURSE multiplicity
zeroTo1;
end association;

association SECT_FAC is
  role SECT_TEACHING__FAC_PERSON_SSNtoFAC_PERSON_SSN : SECT
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSECT_TEACHING__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_FAC is
  role STUD_RADVISES__FAC_PERSON_SSNtoFAC_PERSON_SSN : STUD
multiplicity zeroToN;
  role FAC_PERSON_SSNtoSTUD_RADVISES__FAC_PERSON_SSN : FAC multiplicity
zeroTo1;
end association;

association STUD_GRADCLASS is
  role STUD_MEMBEROF__GRADCLASS_DESIGtoGRADCLASS_DESIG : STUD
multiplicity zeroToN;
  role GRADCLASS_DESIGtoSTUD_MEMBEROF__GRADCLASS_DESIG : GRADCLASS
multiplicity zeroTo1;
end association;

association STUD_PERSON is
  role STUD_PERSON_SSNtoPERSON_SSN : STUD multiplicity zeroTo1;
  role PERSON_SSNtoSTUD_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;

association OFFRNG_QUAR is
  role OFFRNG_QUAR_QNAMEtoQUAR_QNAME : OFFRNG multiplicity oneTo1;
  role QUAR_QNAMEtoOFFRNG_QUAR_QNAME : QUAR multiplicity zeroToN;
end association;

association ASSIGNEDAO_STUD is
  role ASSIGNEDAO_STUD_PERSON_SSNtoSTUD_PERSON_SSN : ASSIGNEDAO
multiplicity zeroTo1;
  role STUD_PERSON_SSNtoASSIGNEDAO_STUD_PERSON_SSN : STUD multiplicity
zeroTo1;
end association;

association ASSIGNEDAO_SECT is
```

```
  role
ASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTIT
LE : ASSIGNEDAO multiplicity zeroTo1;
  role
SECT_TAS__OFFRNG_COURSE_CTITLEtoASSIGNEDAO_SECT_TAS__OFFRNG_COURSE_CTIT
LE : SECT multiplicity zeroTo1;
end association;

association SECT_OFFRNG is
  role SECT_TAS__OFFRNG_COURSE_CTITLEtoOFFRNG_COURSE_CTITLE : SECT
multiplicity zeroTo1;
  role OFFRNG_COURSE_CTITLEtoSECT_TAS__OFFRNG_COURSE_CTITLE : OFFRNG
multiplicity zeroTo1;
end association;

//*---Interactions---*

association BICYCLE_PERSON is
  role BICYCLE_OWNS__PERSON_SSNtoPERSON_SSN : BICYCLE multiplicity
zeroToN;
  role PERSON_SSNtoBICYCLE_OWNS__PERSON_SSN : PERSON multiplicity
zeroTo1;
end association;

association FAC_PERSON is
  role FAC_PERSON_SSNtoPERSON_SSN : FAC multiplicity zeroTo1;
  role PERSON_SSNtoFAC_PERSON_SSN : PERSON multiplicity zeroTo1;
end association;
end package;

//*---END OF OUTPUT---*
```

Bibliography

[1] Elmagarmid, Ahmed; Rusinkiewicz, Marek and Sheth, Amit, editors *"Management of Heterogeneous and Autonomous Database Systems"*, Morgan Kaufmann, 1999

[2] Noe, Penelope *"A Structured Approach to Software Tool Integration"*, MS Thesis, AFIT/GCS/ENG/99M-14, Graduate School of Engineering, Air Force Institute of Technology (AU), 1999, ADA361674

[3] Colonese, Emilia *"Methodology for Integrating the Scenario Databases of Simulation Systems"*, MS Thesis, AFIT/GCS/ENG/99J-03, Graduate School of Engineering, Air Force Institute of Technology (AU), 1999, ADA364951

[4] (DRAFT) *"Reverse Engineering for Data Integration and Sharing (REDIS) Methododlogy Manual"*, C4I Analysis and Database Branch, C4I Test Division, Electronic Proving Ground, Fort Huachuca, AZ, May 1996

[5] Sull, Wonhee and Kashyap, Rangasami L. *"A Self-Organizing Knowledge Representation Scheme for Extensible Heterogeneous Information Environment"* IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No. 2, 5 Apr 1992

[6] Flavin, Matt, *"Fundamental Concepts of Information Modeling"* Yourdon Press, 1981

[7] Kim, *Won,* "Modern Database Systems: The Object Model, Interoperability and Beyond", Addison-Wesley, 1995, ch 25

[8] Batini, C., Lenzerini, M., and Navathe, S. B., *"A Comparative Analysis of Methodologies for Database Schema Integration"* ACM Computing Surveys, Vol. 18, No. 4, Dec 1986

[9] Widom, Jennifer, *"Changes, Consistency and Configuration in Heterogeneous, Distributed Systems"* Stanford University, AFRL-IF-RS-TR-1999-102, May 1999

[10] Arens, Yigal, *"SIMS II: Single Interface to Multiple Sources of Incomplete Information"* USC Information Sciences Institute, AFRL-IF-RS-TR-1998-153, Jul 1998

[11] Chu, Wesley, *"Scalable and Extensible Cooperative Information Systems"* UCLA, AFRL-IF-RS-TR-1999-166, Aug 1999

[12] Buckwalter, Stephen *"Storage/Retrieval Code from Object_Oriented Specifications"*, MS Thesis, AFIT/GCS/ENG/00M-02, Graduate School of Engineering, Air Force Institute of Technology (AU), 2000

[13] Koch, George and Loney, Kevin *"ORACLE, the Complete Reference"*, McGraw-Hill, Oracle Press; 1997

[14] Fausto *"How to Execute a Unix Command from a PLSQL Program?"* http://support.oracle.com, 18 Oct 1999, 13:50:41

[15] Cornn, Gary L. Jr. *"An Object-Oriented Repository-based Software Synthesis System"*, MS Thesis, AFIT/GCS/ENG/00M-05, Graduate School of Engineering, Air Force Institute of Technology (AU), 2000

[16] Ashby, Michael R. *"Tool-Based Integration and Code Generation of Object Models"*, MS Thesis, AFIT/GE/ENG/00M-02, Graduate School of Engineering, Air Force Institute of Technology (AU), 2000

[17] Thomson, Steven A. *"Validation and Verification of Formal Specifications in Object-Oriented Software Engineering"*, MS Thesis, AFIT/GE/ENG/00S-01, Graduate School of Engineering, Air Force Institute of Technology (AU), (Will be published as an AFIT Thesis) 2000

[18] Gunsch, Jason A. *"CLASPICS Documentation"*, Graduate School of Engineering, Air Force Institute of Technology (AU), 1999, unpublished

[19] *"IBM Visual Age for Java Version 2.0"*, http://www.software.ibm.com/ad/vajava; IBM Corporation, 2 Oct 1998

[20] Ashby, Michael R.. *"Tool-Based Integration and Code Generation of Object Models"*, MS Thesis, AFIT/GE/ENG/00M-02, Graduate School of Engineering, Air Force Institute of Technology (AU), 2000

[21] Graham, Robert P., Jr. *"Common Object-Oriented Imperative Language Language Reference Manual"*, Air Force Institute of Technology (AU), Sep 1999, unpublished

[22]Schildt, Herbert, and O'Neil Joe *"Java Programmer's Reference"*, McGraw Hill, 1997

[23] Hamilton, Graham; Cattell, Rick and Fisher, Maydene *"JDBC™ Database Access With Java™"* Addison-Wesley, 1997

[24] Ayers, Danny; Bergsten, Hans; et al. *"Professional Java Server Programming"* WROX Press, 1999

[25] Larson, J., Navathe, S. B.,and El-Masri, R. *"A Theory of Attribute Equivalence and its Applications to Schema Integration"* IEEE Transactions on Software Engineering, Vol. 15, No. 4, 449-425, Apr 1989

[26] Ramesh, V. and Ram, S. *"A Methodology for Interschema Relationship Identification in Heterogeneous Databases"* Proceedings of the Hawaii International Conference on Systems and Sciences, 263-272, 1995

[27] Song, W., Johannesson, P. and Bubenko, J. *"Semantic similarity Relations in Schema Integration"* Proceedings of the 11th International Conference on the Entity-Relationship Approach, 97-120, 1992

[28] El-Masri, R. and Navathe, S. B. *"Object Integration in Logical Database Design"* IEEE Transactions on Data Engineering, 426-433, 1984

[29] DeSouza, J. *"SIS—A Schema Integration System"* Proceedings of the Fifth British Conference on Databases, 167-185, 1986

Vita

Captain Joe Pearson was born in 1965 in Swormville, New York, the sixth of thirteen children. He graduated from Rio Mesa High School, Camarillo, California in 1983, then immediately enlisted in the Air Force as an Electronic and Computer Switching Systems Specialist. In eleven years of enlisted service he worked with a variety of C4I systems around the globe, from AWACS in Saudi Arabia to space systems at Peterson AFB. He completed a Bachelor of Science degree in Computer Science from Regis University, Colorado in 1994, Summa cum Laude. After graduating from Officer Training School, he was assigned to the Defense Information Systems Agency's Joint Interoperability and Engineering Organization in Arlington, Virginia where he continued to work with C4I systems. After three six-month tours deployed in support of NATO operations in the Balkans, he was selected to attend AFIT. While at AFIT, he was reunited with his three daughters, Antraneva, Charlene and Monica, who eagerly anticipate their geographically fortunate follow-on assignment to Colorado Springs.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2000	Master's Thesis

4. TITLE AND SUBTITLE

AN IMPROVED ALGORITHM FOR TRANSLATING RELATIONAL SCHEMAS INTO AN OBJECT MODEL

5. FUNDING NUMBERS

EN

6. AUTHOR(S)

Joseph C. Pearson, Captain, USAF

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 P Street, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/00J-04

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA/ISO
Attn: Doug Dyer, Major, USAF
3701 N. Fairfax Drive
Arlington, VA 22203-1714 (703)696-2203

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Dr Thomas C. Hartrum, ENG, DSN: 785-3636 x4581

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

ABSTRACT (*Maximum 200 Words*)

Today's warfighter is inundated with data from numerous Command, Control, Communications and Computers and Intelligence systems. Integration of these systems is desirable, yet integration results in a static solution to a dynamic problem—by the time a global schema can be devised, it's out of date. Automating schema integration will mitigate this problem, but data model disparity must be addressed via translation to a common data model prior to integration. To address this requirement, this thesis presents an improved, relational to object-oriented schema translation algorithm, which is derived from a base algorithm proposed by another research effort. The improved algorithm incorporates many benefits over the base algorithm, including increased automation, semantic equivalence assurance via tracing key migrations, and decreased inter-entity association redundancy. The improved algorithm is implemented in a highly automated schema translation application, which is demonstrated against a sample schema and validated on another schema from an operational course scheduling system. Given a relational schema, the application provides an initial object-oriented translation and re-translates when schema change is detected.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Schema Translation, Relational to OO Schema Translation Algorithm, Automated Schema Translation Algorithm	139
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL